PDBIS: a unified database end-user interface

Al-Rawi, Ismail A. A.

Award date:
1986

Awarding institution:
University of Bath

PDBIS: A UNIFIED DATABASE END-USER INTERFACE

submitted by Ismail A. A. Al-Rawi
for the degree of Ph.D.
of the University of Bath
1986

I. Al-Rawi

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

To

The Glory of God

And

The Memory of My Father

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to all who have assisted me during my research and in the preparation of this thesis. Particular thanks are due to my supervisor Professor J. P. Fitch for taking over the supervision and for his helpful advice and encouragement. I also thank Dr. T. J. King who continued to give me his support and encouragement even after he left the University.

My thanks go also to the Iraqi government for the scholarship they provided.

SUMMARY

The growth of computer technology and the acceptance of computer methods in simple daily life depends largely on the successful establishment of effective man-machine communications. Therefore in recent years, the user has become the prime factor of system design. The terminals instead of being peripheral devices, have become the mirror which reflect the whole service.

The current development in data processing technology and the current revolution in office information systems, requires a very efficient database end user interface capable of satisfying the increasing requirements by all classes of end user. Studies in this respect have shown that the graphical specification for such an interface are more successful than the linear and the natural language specifications.

In this thesis, PDBIS (Perq DataBase Interface System) design and implementation are presented. It is an easy to use and easy to learn graphical database end user interface, running on the Perq machine to provides a unified facility for accessing databases managed and controlled by different database management systems operating in different machines. PDBIS syntax is very wide and covers most of the manipulation and the updating operations that are required by most relational database management systems. The syntax and the database structure are presented to the user on the Perq screen in a very natural form, so that the user can formulate his query easily depending on the menu-selection technique, and assisted by the instruction statements and the different feedback methods. PDBIS provides the user with the facility to reformat the accessed data, and for report generation as well. Using PDBIS in a distributed database environment is also described.

Thesis Overview

The reader of this thesis is considered to have little knowledge about database technology; therefore the thesis is structured in such a way that each chapter provides the background for the following one. Chapter 1 presents the main topics of the database concept emphasising the relational model, as this is more related to this research topic than any other database model. To complete the overview on the relational database, a survey of the well known relational DBMSs is given.

To provide the background for the topic of this research, chapter 2 presents the concept of an end user interface in general and the relational database query language in particular. The end users are then classified and the casual user is discussed in detail. A view of the formal high level relational language is given, and the concept of easy to use and user friendly terms are explained as a preparation to define the query language design considerations.

Chapter 3 becomes more specific by reviewing a number of related graphical database interfaces, particularly QBE, FORAL-LP, and CUPID. These languages are explained and discussed in detail together with the experimental studies of others which have been carried out on them to measure their suitability for use by the casual user. A conclusion is then given which summarizes the positive and the negative aspects of these languages, so that they can be considered as a feedback in the PDBIS design.

PDBIS is introduced in chapter 4. A number of justification points for using the graphics technique in PDBIS are given. An overview of the system is presented, and the main objectives and properties of PDBIS are defined. The chapter presents the hardware and the software (the Perq and the graphics utility) specification on which PDBIS is implemented. At the end of the chapter the PDBIS screen layout design considerations are discussed.

The configuration of PDBIS is given in chapter 5. PDBIS components and the operations flow among these components are explained. Before the specifications of the screen manager program, the controller program, the analyzer program, and the output file manager program are discussed, the query formulation process and the entire set of PDBIS syntax are described. The general structure of the analyzer is given, followed by detailed explanation of the SQL analyzer and its implementation supported by a variety of examples.

The use of PDBIS as a unified interface in a distributed database environment is presented as a further work in chapter 6. The design and the description of the distributed database system, D-PDBIS, in which PDBIS is used as a unified end user interface are also presented in this chapter.

The conclusion of the thesis is given in chapter 7.

Four appendices are also included. Appendices A-C explain in detail, with examples, the implementation of PDBIS with CODD, DBASE-II, and LINUS respectively.

Appendix D presents the specification of the Output File Manager (OFM) with examples of how it would be used.

A number of photographs of selected examples are also included in the thesis.

CONTENTS

DEDICATION		ii
ACKNOWLEDGEMENTS		iii
SUMMARY		iv
THESIS OVERVIEW		v

CHAPTER 1: INTRODUCTION TO DATABASES

1.1.	Database concepts	1
1.2.	DBMS (Database Management System)	2
1.3.	Requirements of a DBMS	3
1.4.	Database architecture	3
1.5.	Data Models	4
1.5.1.	Hierarchical model	5
1.5.2.	Set (network) model	6
1.5.3.	Relational data model	7
1.6.	Normalization of relations	11
1.7.	Data languages	14
1.7.1.	Algebraic approach	15
1.7.2.	Predicate calculus approach	18
1.7.3.	Comparison	19
1.8.	Survey of relational database systems	20

CHAPTER 2: USER INTERFACE

2.1.	End-User Interface	27
2.2.	A relational database query interface	28
2.3.	User classification	29
2.3.1.	Casual users	30
2.3.2.	Non-programmer professionals	34
2.3.3.	Conventional computer application programmers	35
2.3.4.	Database systems personnel	35
2.4.	Formal high level relational language	35
2.5.	Ease-of-Use and User-Friendliness	37
2.6.	Language design considerations	39

CHAPTER 3: GRAPHICAL QUERY LANGUAGES

3.1.	Introduction	41
3.2.	Query-By-Example	42
3.2.1.	Experimental study	48
3.2.2.	Summary	50
3.3.	FORAL-LP	51
3.3.1.	Summary	56
3.4.	CUPID	58
3.4.1.	Experimental study	63
3.4.2.	Summary	64

3.5. Conclusion 64
3.6. Spatial and Iconic Systems 65

CHAPTER 4: PDBIS: Perq Database Interface System

4.1. Introduction 68
4.2. System overview 69
4.3. Interface objectives 74
4.4. Perq 76
4.5. Graphics facilities 77
4.6. Screen design 78

CHAPTER 5: PDBIS SPECIFICATION

5.1. PDBIS components 83
5.2. System layout and operations flow 85
5.3. Query formulation 86
5.4. Program specification 91
5.4.1. Screen-Manager 91
5.4.2. Controller 92
5.4.3. Analyzer 93
5.4.4. Output-File-Manager 98
5.5. SQL analyzer 99
5.6. PDBIS-SQL Interface 106
5.6.1. Retrieving operations 107
5.6.2. Update operations 116

CHAPTER 6: FURTHER WORK: PDBIS IN A DISTRIBUTED ENVIRONMENT

6.1. Distributed database system 121
6.2. D-PDBIS 123
6.2.1. D-PDBIS overview 123
6.2.2. Design objectives 126
6.2.3. System architecture 126
6.2.4. Components architecture 129
6.2.5. The query processing 131

CHAPTER 7: CONCLUSION

7.1. End User Interface 134
7.2. PDBIS 135

APPENDIX A: CODD INTERFACE

1. SALT analyzer 139
2. PDBIS-SALT Interface 140

APPENDIX B: DBASE-II INTERFACE

1.	DBASE-II analyzer	146
2.	PDBIS-DBASE-II Interface	147
2.1.	Retrieving operations	147
2.2.	Updating operations	151

APPENDIX C: LINUS INTERFACE

1.	LILA analyzer	154
2.	PDBIS-LILA Interface	155
2.1.	Retrieving operations	155
2.2.	Update operations	160

APPENDIX D: Output-File-Manager (OFM)

1.	OFM specification	162
1.1.	Formatted output	162
1.2.	Report generation	165
1.2.1.	General formats of a report description	166
1.2.2.	General rules	171
1.3.	Graph display	175

REFERENCES 177

CHAPTER 1

INTRODUCTION TO DATABASES

1.1. Database concepts

Until the late 1960's the philosophy of computerized information systems was to provide data for specific applications. The same data was frequently being resorted, reformatted and duplicated for different applications. This approach to data organization by physically repositioning it with other data produced process oriented files. As a result of the great advance in computer resources such as storage media, processing speed and communication management, the term *database* became current, and is used to indicate the collection of interrelated data stored together without unnecessary redundancy to serve multiple applications. The data are stored so that they are independent of the program which uses them. A unified and controlled approach is used in adding new data and modifying and retrieving existing data within the database to provide the management with the necessary information for the decision making process.

Databases have become one of the most rapidly growing areas of computer and information science. In less than twenty years, database systems have come from nothing to be a major topic of current interest. As the employment of better data management software spreads, characteristics of databases such as data independence, control redundancy, interconnectiveness, and security protection have spread with it. Top management of major enterprises have grown to appreciate the importance of their database as their organizations become critically dependent on the successful operation of a database. The essential objective of database systems is to make applications development easier, cheaper, faster and more flexible, and to provide an enterprise with centralized control of its operational data.

1.2. DBMS (Database Management System)

The database management system is another step in the natural development of the computerized information systems. As occurred previously in other cases of development in computer systems, such as peripheral handling, filestore management, and high level languages, *etc.*, the operations which are frequently required have been collected out of the user program and incorporated into a generalized supporting system. The DBMS is a software package which represents a generalized tool for manipulating large databases and controlling access to the physical representation of data, and shielding the database users from the hardware level. Its interfaces generally provide a broad range of languages to aid all sorts of users.

To meet the various objectives of the database, the DBMS must support a variety of functions (KIM-79) such as:

(1) An efficient optimisation process to meet the response-time requirements of interactive users.

(2) Efficient file structures definition in which to store the database and efficient access paths to the stored database.

(3) A high-level, non-procedural data language which provides the following capabilities for users; query, data manipulation, data definition and data control.

(4) To provide views and snapshots of the stored database.

(5) To provide integrity control; validation of semantic constraints on the database during data manipulation and rejection of the invalid data manipulation statements.

(6) Concurrency control; synchronisation of simultaneous updates to a shared database by multiple users.

(7) Selective access control; authorisation of access privileges to one user's database to other users, and also to allow only authorised processes to access the data.

(8) Recovery from both soft and hard crashes.

(9) A report generator for the results of interactions against the database and such application-oriented computational facilities such as statistical analysis and graphics display.

(10) Monitoring tool, to allow the database administrator to control the performance of the database and to adjust things.

1.3. Requirements of a DBMS

Many enterprises now rely upon computer database management systems. The database itself can become an extremely valuable resource and therefore every effort must be made to maintain its viability. It should be possible to transport the data from one computer installation to another; to utilise new developments in computer technology; to reconfigure an existing computer system with minimal effect upon the database management system, in other words, to operate the DBMS independently from the main computer operating system; to restructure the database to optimise access mechanisms; to conserve the validity of the contents of a database; to protect the database from access by unauthorised personnel; and incorporate evolutionary changes to the database model. These requirements must be borne in mind when designing a DBMS. It should also be considered in designing the data sublanguage or any end-user interface like PDBIS (the main topic of this thesis), so that any detrimental effect on the DBMS can be avoided.

1.4. Database architecture

The CODASYL Data Base Task Group (DBTG) has had a considerable influence on the evolution of the DBMS concepts. The DBTG produced DBMS specifications in 1969

(DBTG-69) and in 1971 (DBTG-71). Their specifications became a basis for several major DBMS products, eg. IDS-II, IDMS, DMS.

The CODASYL specifications, enhanced by more recent work, have gained wide acceptance (CODA-78). A further major input to the DBMS concept was provided by the ANSI/SPARK report, published in 1975 (ANSI-75). This report defined three general levels of database modelling and description: the conceptual level, which is the description of the information modelled in the database; the external level, which consists of the user view of the database, and the internal level, which describes the physical layout of the data. The data description at the conceptual level should not contain references to the way the data is stored and accessed, but describe only the fundamental elements and properties of the information model.

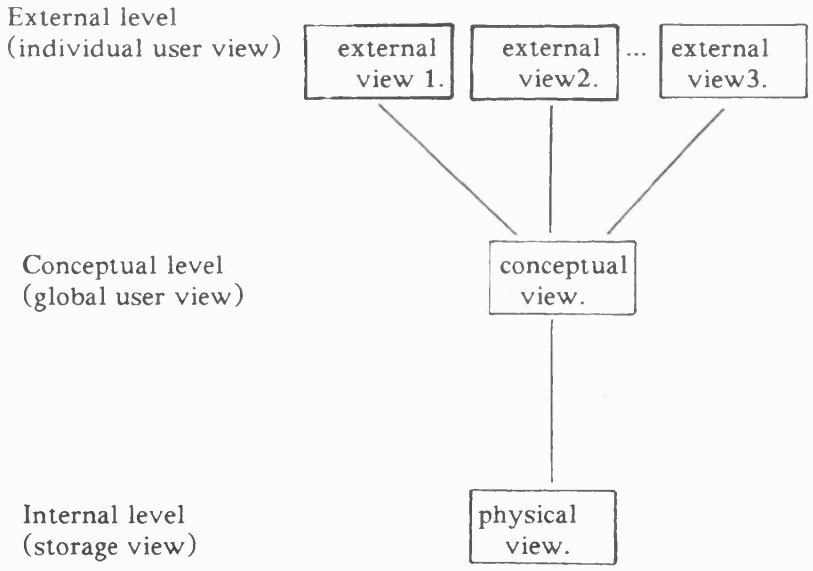


Figure 1.1 The three levels of data architecture.

1.5. Data Models

In the database context a data model is generally understood to mean any formally definable class of data structure which can be used as the basis for the

development of a data processing application. Such a structure, which shows the logical association between data items, resembles, to a certain extent, the real world situation that has to be manipulated. This logical structure of the data is the main issue of the design and operation of a DBMS. At the present time, most commercially available DBMS support one of the three following major models:

(1) Hierarchic (tree structure) model.

(2) Set (Network structure) model.

(3) Relational model.

1.5.1. Hierarchical model

The hierarchical model forms the basis of IBM's IMS, and other significant database products. This data model can be described with the aid of hierarchical definition trees, which specify record types and entity types, and the relationships between them. Record types are grouped into 'levels'. The highest level of the hierarchy has only one record type called the root. All record types except the root record type are related to only one record type at a higher level, and this is called the parent. No record type can have more than one parent. Each record type can have one or more record type related to it at a lower level (one to many relationship).

An example of hierarchical structure is given in Figure 1.2.

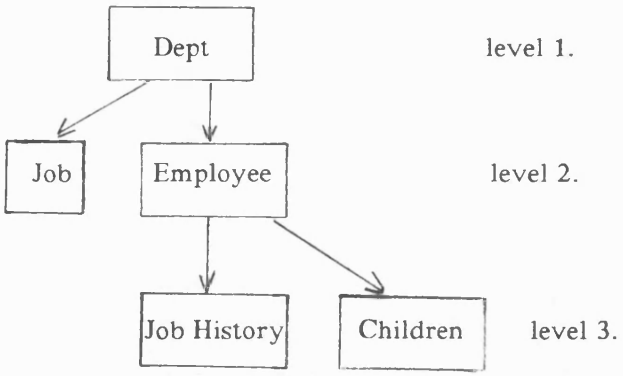


Figure 1.2

The applications programmer may retrieve data using calls to the DBMS from a programming language, navigating through the hierarchical structure one record at a time. Some high level query languages may be used to manipulate data in hierarchical structures as in system 2000 (FRY-76) and Query-By-Example (ZLOO-76) without writing programs.

1.5.2. Set (network) model

The data structure set (network) data model formed the basis of the 1971 CODASYL DBTG proposals. As in the hierarchical structure, data is represented by records and links (relationship). However, a network is a more general structure than a hierarchy because a given record occurrence may have any number of immediate superiors as well as any number of immediate dependents (many-to-many relationship). These many-to-many relationships have to be decomposed into one to many relationships which are then directly modelled as in the hierarchical structure; this is done by introducing a third record type called the connector. A connector represents the association between the first two record types. An example of network structure is given in Figure 1.3.

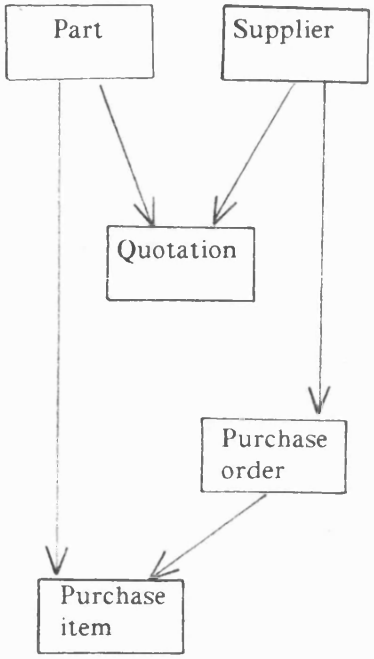


Figure 1.3

In the above figure the relationship between Supplier and Part record type is many-to-many; it has been decomposed into a one to many relationship with the associated record type Quotation. The connector Quotation may or may not contain any data value.

The CODASYL specifications for data retrieval depend on using a data manipulation language embedded within a host programming language like Cobol or Fortran. The application programmer navigates through the database structure using one record at a time, as in the case of hierarchical structure.

1.5.3. Relational data model

Since PDBIS (the subject of chapter 4) is a database end-user interface, which basically supports the relational model, I shall discuss this model in a little more detail through the rest of this chapter.

The considerable interest in relations as a data model for DBMS initiated from a paper by Codd published in 1970 (CODD-70) which gave a solid definition for relations in a database context, emphasizing their advantages for data independent and symmetry of access. Data is modelled as n-ary relations, which can be represented as two dimensional arrays (tables). Each row of the table represents one tuple of the relation. The number of tuples in a relation is called the cardinality of the relation. Each column of the table represents one domain. The number of domains in the relation is known as the degree of the relation, the table itself represents an entity class, and a collection of tables (relations) represent the database.

Dept

Dept#	Depname	Manager	Loc

Employee

Emp#	Empname	Dept#	Sex	Salary

Job

Dept#	Job#	Job-Description

Job History

Emp#	Job-Date	Title

Children

Emp#	Child-name	Age	Sex

Figure 1.4 Relational model.

Each table (relation) in the relational model represented in Figure 1.4 should have the following properties:

(1) The items in any selected column are all of the same kind, whereas items in different columns need not to be of the same kind.

(2) Each item in any columns should be a simple number or character string (not a set of values or repeating group).

(1) Ordering of rows within a table is immaterial.

(2) Duplication of rows are not allowed in the table.

(3) Ordering of columns within a table is immaterial.

(4) Each column within a table is assigned a distinct name.

PART#	PNAME	COLOR	WEIGHT	CITY
P1	Cog	Red	10	London
P2	Screw	Blue	20	Paris
P3	Nut	Red	14	Rome
P4	Bolt	Green	20	London
P5	Cog	Blue	10	Athens

Figure 1.5 Relation part.

Figure 1.5 shows a relation containing information about parts, in which each column name corresponds to an domain of the relation, and each row to a tuple. Normally, the database consist of several such relations. This approach of data modelling is based on the mathematical theory of relations, and most of the terms being taken from this branch of mathematics (MART-77). From the mathematical notation based on relational algebra or relational calculus for describing relations and operations between them, Codd has devised a mathematical sublanguage for manipulating such a database (CODD-71a). A conventional high level language could be translated into this sublanguage to interact with relations in terms of high level query language. The first of these was Codd's 'relational algebra' which in addition to the set operators of union, intersection and difference, contained join, projection, and selection (CODD-71b). By these operations new relations can be constructed with different degrees and cardinalities to provide a different set of data items for different users according to their views and requirements. These operations give a degree of flexibility in manipulating data and extending the data structure. This flexibility is not possible with hierarchical and network models. The user view of the data can thus consist of sets of two dimensional tables with operations for extracting columns and

joining them. "Both the application programmer and the interactive user view the database as a time-varying collection of normalized relations of assorted degrees" (CODD-71c).

Relations may also be thought of as highly disciplined conventional files (DATE-81) where a relation resembles a file, a tuple a record (occurrences not type) and attribute a field (type not occurrences). These relational files are distinguished from the traditional, undisciplined files by the following features:

(1) Each file contains only one record type.

(2) Every record occurrence in a given file has the same number of fields.

(3) Each record occurrence has a unique identifier.

(4) Within a file (relation) records either have an unspecified ordering or are ordered according to values contained within those occurrences (the ordering field is not necessarily the primary key).

These characteristics of the relational data type can be easily conceived by the user, and help to design easy to use relational database languages. In addition this model contains such well known important features like:

(1) A high degree of data independence.

(2) It operates on one set (relation) at a time rather than on one record at a time.

(3) It provides an environment in which it is easy to create and modify the database.

(4) It allows the user dynamically to embed integrity and authority statements (independent of the underlying structure) in the systems.

(5) Can be understood easily by users with no training in programming.

These features and characteristics give the relational model the superiority over the other data models.

1.6. Normalization of relations

The normalization process of relations is a very important aspect of relational database design. This topic helps the designer to find a suitable logical structure for the data; in other words, how he decides what relations should be involved and what attributes should be contained. The normalized data structures will minimize the probability of future disruption.

"Normalization is a step-by-step reversible process of replacing a given collections of relations by successive collections in which the relations have a progressively simpler and more regular structure. The simplification process guarantees the recovery of the original collection of relations and there is no loss of any information" (CODD-71d). The main objectives of normalization as stated by Codd are:

(1) To make it possible to tabulate any relation in the database; that is in particular to maintain the simplicity of the data item in any tuple and avoid any set of values or repeating group.

(2) To obtain a powerful retrieval capability by means of a simpler collection of relational operations than would otherwise be necessary.

(3) To free the collection of relations from undesirable insertion, update and deletion dependencies.

(4) To reduce the need for restructuring the collection of relations as new types of data are introduced, and thus increase the life span of application programs.

(5) To make the relational model more informative to users.

(6) To make the collection of relations natural to the query statistics where these statistics are liable to change as time goes by.

The normalization process is built around the concept of normal forms. Five distinct normal forms can be recognized (DATE-81): first, second, third, fourth, and

fifth normal form (abbreviated as 1NF, 2NF, 3NF, 4NF, and 5NF respectively). Each of them has a certain specified set of constraints to be satisfied by the relation in order to be in that normal form.

The first three normal forms are well defined. Relations in 3NF is a reasonable strategy for the designer to achieve. Usually a relation in 3NF will have already satisfied the 1NF and 2NF constraints.

In 1NF, the relation domains contains only atomic values (no repeating group or decomposable values).

Figure 1.6 represent a 1NF relation with a primary key (S#, P#).

R

S#	P#	CITY	STATUS	QTY

Figure 1.6

A relation R is in 2NF if and only if it is in 1NF and every nonprime attribute (nonkey) is fully functionally dependent on the primary key.

This definition shows that the primary key can be a combination of two or more attributes. Then the relation that is in 1NF and not in 2NF must have a composite primary key, and such a relation can be split to an equivalent collection of 2NF relations.

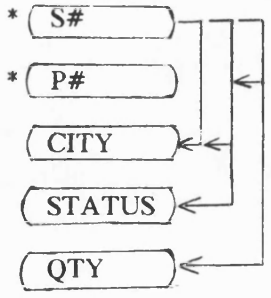


Figure 1.7 Functional dependencies in the relation R.

The asterisks indicates the members of the primary key.

Figure 1.7 shows that STATUS and CITY are not fully functionally dependent on the primary key, and they are not mutually independent. Therefore relation R is not in 2NF and it has to be split into two 2NF relations as shown in Figure 1.8.

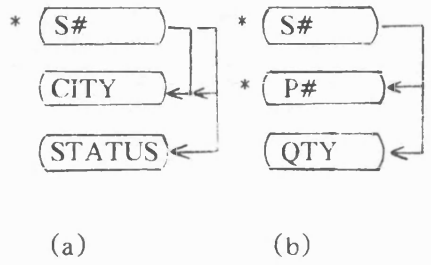


Figure 1.8 Two 2NF relations.

A relation R is in 3NF if and only if it is in 2NF and every nonprime attribute (nonkey) of R is nontransitively dependent on the primary key of R. For example, the relation in Figure 1.8 (a) is not in 3NF since status is transitively dependent on S#.

Conversion to 3NF removes these transitive dependencies by decomposing the relation into two 3NF relations, as in Figure 1.9, without any loss of information, as the original relation can be composed by applying the natural join on these 3NF relations over attribute CITY.

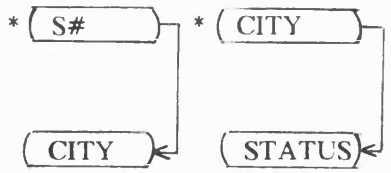


Figure 1.9 3NF relations.

In recent work, the 3NF is redefined by Codd and Boyce (CODD-74), to be more general and more restrictive, and it has been termed as BCNF.

A complete description of this topic can be found in (CODD-71c, CODD-71d, DATE-81).

1.7. Data languages

Access to the database is provided by a data manipulation language (DML), which is a set of operations that permit access to the data as organized by a data model. The data manipulation language operations fall into two categories: retrieval and updating. Retrieval operations are concerned with obtaining the data values stored in the database. Updating operations are concerned with changing the data values or data relationship of a database. The DML languages which are mainly concerned with data are called data sublanguages (CODD-71b). A data sublanguage may be embedded in a general purpose programming language, or it may be stand alone, in which case it is commonly called a query language. A query language enables the user to formulate his requests with a simple language. Even though they are called query languages they also contain simple updating operations.

Data manipulation languages can be either generalized or application oriented. Generalized data languages have general commands for data manipulation without concern to any particular application. Application oriented data manipulation languages follow closely the concept and terminology used in a particular application. Data manipulation languages span a wide range of procedurality. At one end there are more procedural data languages, in which one has to specify what is wanted as well as how to obtain it. On the other end there are less procedural data languages, in which the programmer need only specify what is wanted and the system decides how to obtain it. The more procedural data languages are simpler to implement, since the programmer directs the database management system step by step on how to navigate through the structure and get the required data. The CODASYL and the hierarchical database manipulation languages such as IDMS-DML, IDS-II/DML, IMS-DL1, *etc.*

belong to this category of data manipulation language. The less procedural data languages are complicated to implement, but allow the system to decide how to get the data. The nonprocedural data languages are designed mainly for non-programmers, "casual users", who want to use the database, but who do not want to know anything about computers. This category of data manipulation language includes most of the relational data sublanguages such as SQL, CUEL, QBE, *etc.*

In the relational model, data sublanguages break down into two broad classes of approaches, the algebraic approach and predicate calculus approach.

1.7.1. Algebraic approach

This approach provides a collection of operations on relations of all degrees suitable for selecting data from a normalized relational database, yielding new relations as a result. In addition to the main set operators (union, intersection, difference, and cartesian product) the algebraic sublanguage include special operators which are selection, projection, join, and division (CODD-71b).

UNION: The union of two relations R1 and R2, is a set of all tuples T, belonging to either R1 or R2 or both. The union operator is applied to relations of the same degree which have compatible domains.

$R1 \cup R2 = \{ T \mid T \in R1 \cap T \in R2 \}$.

$\cap$ is an inclusive or.

Example: $\{1,2,3\} \cup \{1,2,4\} = \{1,2,3,4\}$.

INTERSECTION: The intersection of two relations R1 and R2, is the set of all tuples which belong to both relations. R1 and R2 must have the same degree and take their values from compatible domains.

$R1 \hat{} R2 = \{ T \mid T \in R1 \hat{} T \in R2 \}$.

$\hat{}$ is a conjunction and.

Example: {1,2,3} $\hat{}$ {1,2,4} = {1,2}.

DIFFERENCE: The difference of two relations R1 and R2 of degrees d1 and d2 is the set of tuples T, belonging to R1 and not to R2. R1 and R2 must have compatible domains and the same degree.

$R1 - R2 = \{ T \mid T \in R1 \hat{} T \notin R2 \}$.

Examples: {1,2,3} - {2,3,4} = {1}.

{2,3,4} - {1,2,3} = {4}.

CARTESIAN PRODUCT: The cartesian product of two relations R1 and R2 of degrees d1 and d2, is the set of tuples T, whose first d1 components are from a tuple in R1, and whose last d2 components from a tuple in R2. If $(r1_1 \ldots . r1_{d1})$ is a tuple in R1 and $(r2_1 \ldots . r2_{d2})$ is a tuple in R2 the concatenation between them yield the tuple $T=(r1_1 \ldots r1_{d1}, r2_{d1+1} \ldots r2_{d1+d2})$.

$R1 \otimes R2 = \{ (r1,r2) \mid r1 \in R1 \hat{} r2 \in R2 \}$.

Example: {1,2} $\otimes$ {a,b,c} = {(1,a),(1,b),(1,c),(2,a),(2,b),(2,c)}.

SELECTION: This operation yields a 'horizontal' subset of a given relation, that is a subset of tuples within the given relation for which a specified predicate is satisfied.

$R[I \varnothing J] = \{ r \mid r \in R \hat{} (r[I] \varnothing r[J]) \}$.

Where I and J are attributes values in R, and $\varnothing$ is one of the comparison operators; =, $\neq$, >, <, $\geq$, and $\leq$.

Example: R(a1 a2 a3)

a	2	1
b	1	4
c	3	3
d	4	5

R[a2 = a3] = r(a1 a2 a3)
c 3 3

PROJECTION: This operation takes a subset of domains of a relation and forms a new relation.

R[A] = { r[A] | r ∈ R }.

where A is a list of attributes for R, r[A] is a tuple derived from r[R], consisting only of those attributes identified by A.

Example: R(a1 a2 a3) R[a1] R[a2] R[a1 a2]
a 1 f a 1 a 1
b 3 g b 3 b 3
c 3 n c 2 c 3
d 2 f d d 2

JOIN: The join of two relations R1 and R2 over a subset of domains of the two relations is a new relation R3, which has the concatenated tuples of R1 and R2, with common values in that subset of domains.

R1[I Ø J]R2 = { (r1, r2) | r1 ∈ R1 ^ r2 ∈ R2 ^ (r1[I] Ø r2[J]) }.

(r1, r2) represent m+n degree tuple formed from concatenating r1 and r2.

Example: R1(a b c) R2(d e)
a 1 1 2 u
b 1 2 3 u
a 2 1 4 v
c 2 3
d 3 4

R1[c = d]R2 = (a b c d e)
b 1 2 2 u
c 2 3 3 u
d 3 4 4 v

DIVISION: The division of two relations R1 and R2 of degrees d1+n and d1 is a relation of degree n. The (n+i)th attribute of R1 and the ith attribute of R2 must be defined on the same domain.

This operation is defined by Codd (CODD-71b) as:

Let $g_T(x) = \{ y \mid (x,y) \in T \}$.

$g_T(x)$ is the image set of x under the binary relation T.

Let A be the attributes set in R1 and B is the attributes set in R2.

Let $\overline{A}$ be the attributes set that is complementary to A.

Providing R1[A] and R2[B] are union-compatible, the relational division of R1 on A by R2 on B is defined by:

$R1[A \div B]R2 = \{r1[\overline{A}] \mid r1 \in R1 \hat{} r2[B] \subseteq g_R(r[\overline{A}])\}$.

It can also be defined in terms of joins, differences, and projection as follows:

$R1[A \div B]R2 = R1[\overline{A}] - ((R1[\overline{A}] \otimes R2[B]) - R1)[\overline{A}]$.

For examples, see (CODD-71b).

In fact, intersection, join, and division operation can be expressed in terms of the other set operations such as cartesian product, selection, projection *etc.*, but the availability of these operations in an algebraic language gives direct facilities to the users and simplifies the expression of the queries. For example, join is a cartesian product followed by a select, and then project.

1.7.2. Predicate calculus approach

The idea of using predicate calculus as the basis for data sublanguage appears to have originated in a paper by Kuhns (KUHN-67). The concept of a relational calculus that is an applied predicate calculus specifically tailored to relational database was first proposed by Codd in 1971 (CODD-71b, CODD-71a). A language explicitly based on this calculus called data sublanguage ALPHA (DSL ALPHA), was also presented by Codd (CODD-71a), which has the full power of relational calculus, by applying predicate calculus with tuple variables. DSL ALPHA itself was never implemented,

but a language similar to it in its characteristics called QUEL was used as the query language in the system INGRES (HELD-75, WONG-76).

A fundamental aspect of the calculus of Codd (CODD-71b), and of the data languages based on it, is the notion of the tuple variable. A tuple variable is the variable that ranges over some specified relation, whose only permitted values are tuples of that relation. If tuple variable TV ranges over relation R, then, at any given instant, TV represents some individual tuple of R. For example, the query "Get supplier numbers for supplier in London" can be expressed in QUEL as:

```
RANGE OF SX is SUPPLIER
RETRIEVE (SX.S#) WHERE SX.CITY = 'London'
```

The tuple variable SX range over the relation SUPPLIER. The query may be paraphrased: "For each possible value of the variable SX, retrieve the supplier component of that value, if and only if the city component has the value 'London'".

Lacroix and Pirotte (LACR-77a) have proposed an alternative relational calculus, the domain calculus, in which tuple variables are replaced by domain variables, that is, the variables that range over the underlying domains instead of over relations. A language called ILL based on this calculus is presented by the same authors in 1977 (LACR-77b). Query-By-Example can be considered as an implementation of this type as well (DATE-81).

1.7.3. Comparison

Codd in his paper (CODD-71a) described the relational algebra based data sublanguage as an intermediate level sublanguage which is at a lower level than the sublanguage based on relational calculus, in which the system takes over complete responsibility for searching the database. Codd also characterized the algebraic approach as "constructive", while the calculus approach is "descriptive", providing

optimal search and authorization capability, and transformality when the structure changes in the data model. In addition, he also considers the relational calculus as more close to the natural language than the relational algebra, because requesting data by its properties is far more natural than devising a sequence of operations for its retrieval.

On the other hand, in spite of all the above declarations the algebraic approach is still favored by researchers working in fact retrieval. The algebraic approach provides a yardstick against which other languages can be measured. The algebra also lays a foundation for research into several other aspects of database management, such as database design, view definition, and restructuring.

Many specialists assume that by specifying a sequence of algebraic operations, the user has better control over the query processing, resulting in a better performance than the relational calculus approach. Zloof said that this is not generally the case because of two main reasons (ZLOO-78): first, the user's specified sequence of operations may not generally be the optimal sequence, and thus a special optimizer will be required to map a user specified sequence into a more optimal one; second, processing a query (transaction) that follows the relational algebra operations can be very inefficient and require substantial storage space, (for creating intermediate temporary relations along the way), since intermediate temporary relations may be an order of magnitude larger than the source and target relations. However, many people consider both the data sublanguage based on the relational calculus and the data sublanguage based on relational algebra can be complete, and possesses the same selective power (CODD-71b).

1.8. Survey of relational database systems

Dozens of relational database systems have been implemented since E. F. Codd introduced the relational model of data in a series of pioneering papers between 1970

and 1971 (CODD-70, CODD-71a, CODD-71b, CODD-71c, CODD-71d). In this section some of these systems will be reviewed, so the range of possible variants can be appreciated.

Different levels of implementation of a relational model of data can be distinguished, ranging from the user's view of the information structure to the actual storage structure of data. Two levels are clearly distinguishable; the tuple-by-tuple access level, and the set of tuples (relation) operation level. A number of systems were developed for use as low-level, database access and storage subsystems for implementing high-level, non-procedural, relational data sublanguage such as IBM's RM/XRM, GAMMA-0 (BJOR-73) and ZETA/MINIZ (CZAR-75). At about the same time a number of other prototype systems such as MITs MADAM, IBMs SEQUEL, and INGRES were implemented primarily to demonstrate the feasibility of supporting a high level, non-procedural data language based on the relational algebra or the relational calculus.

MADAM (GOLD-70) was the first relational system to become operational at MIT in 1970, utilizing the large directly addressable virtual memory of MULTICS. A year later, MIT's RDMS system (STEU-74) which was virtually identical to MADAM was developed. The interesting feature of this system is the division of the storage space into what may be called the relation space and the domain space, and the division of the software into a set of procedures which operate on the domain space. The relation space is a set of MULTICS segments used for storage of all the relations in the database. The domain space is also a set of segments of MULTICS used for storage of all of the items in each of the relations. In this system data items are encoded to a fixed-length identifier, and these identifiers are used in the stored relation rather than the actual data items. Using this approach, the stored relations are usually much smaller than the corresponding relations containing the actual data values, and thus

can be manipulated much more easily. RM (LORI-71) is another relational system which adopted similar techniques. It was developed at the IBM Cambridge, Scientific Center, Cambridge, Massachusetts, for use as a database access and storage subsystem. RM was designed to support only binary relations, so its uses are restricted. Therefore, in 1973 XRM (LORI-74) was implemented by extending RM to support higher degree relations. This extension was accomplished by storing the n-ary tuples in RM domain space and indexes in the RM relation space to provide associative access to the n-ary tuples. XRM has been used as a database access and storage subsystem for SEQUEL, GMIS, GXRAM and Query-By-Example.

The first relational system developed by IBM was IS/1 (NOTL-72), which became operational at the IBM UK Scientific Center, Peterlee, England in 1971. Experience with IS/1 led to the implementation of PRTV (TODD-76) in 1973, which had a substantial improvement over IS/1 in storage structure and response time. IS/1-PRTV supported the ISBL data language based on relational algebra. IS/1-PRTV allows the user to extend the capabilities of ISBL by writing PL/1 application programs and linking them to the base system. The most important feature of PRTV is its optimizer. First it transforms an ISBL expression into an algebraically equivalent one which can be more efficiently evaluated (HALL-76). Next it attempts to choose an efficient set of access paths by considering the estimated cost of various alternative traversals.

At General Motors Research, Warren, Michigan, a prototype system called RDMS (WHIT-74) was implemented. This system was one of a few systems that were intended to investigate the possibility of developing a generalized information system which supports a relational model as well as providing facilities for the analysis and display of retrieved data. REGIS (JOYC-76) is the subsequent version of RDMS, which has been developed to be one of the few decision support systems built around relational systems.

At the University of Cambridge CODD (KING-83) a DBMS was designed and implemented to handle a large volume of historical data. It is very much influenced by PRTV, and supported two relational languages, SALT and CHIPS (KING-79), based on relational algebra.

Honeywell has released LINUS (LINU-78) as a generalised DBMS for accessing the centralised MULTICS relational data store (MRDS). It provides a complete database management capability including both retrieval and update operations. Database subsets are selected through a data sublanguage called LILA (LINU-78).

At the IBM Research Laboratory in San Jose, California, SEQUEL (ASTR-75b, ASTR-75a) was implemented to support the SEQUEL data sublanguage (BOYC-74). Experience with the SEQUEL prototype and GAMMA-O provided the basis for the development of system-R (ASTR-76, ASTR-79), which was designed and developed over the period of 1974 to 1979 as a prototype relational system. The implementation of this system has resulted in several outstanding contributions to relational database research. It supports many of the features normally expected in a database system. These include multi-access, integrity controls and user views. The system supports the SEQUEL (later SQL) data language as well as other interfaces. The SQL (CHAM-76) data sublanguge has been considered as one of the few complete successful database query language. System-R has three major components; User Friendly Interface, RDS (Relational Data System), and RSS (Relational Storage System).

At the IBM Thomas J. Watson Laboratory, Yorktown Heights, New York, QUERY-BY-EXAMPLE (ZLOO-75) was implemented to support the novel two dimensional query language QUERY-BY-EXAMPLE (QBE) (ZLOO-77). Development of this system, as in the case of system-R, significantly benefitted from experience with the SEQUEL prototype implementation. The data base language QBE has been

evaluated to be one of the most successful, easy-to-use database sublanguages, and is always compared with the SQL data sublanguage. QBE has influenced most of the recent development of graphical query language including PDBIS. Therefore, details of this query language are given in chapter 3.

The first IBM relational database product derived from system-R was announced in 1981 and called SQL/Data System (SQL/DS)(IBM-81). This system was designed for IBM's intermediate range of computers running on the DOS/VSE operating system environment or processors supported by VSE/Advanced Functions Release 3. SQL/DS interfaces with the CICS data communication subsystem, and used in conjunction with DL/1 database product (CODD-81).

IBM most recent relational database offering is Database 2 (DB2) (HADE-84). This system was developed cooperatively by several IBM branches in USA and England. DB2 is available on MVS/370 or MVS/XA. A query management facility (QMF) is attached to the system to provide interactive users with access to DB2 data and report generation facilities, using either a form of QBE or SQL data sublanguage. Data Extract (DXT), another IBM product, permits data to be transferred from DL/1 database, VSAM files, or physical sequential files to DB2.

Many other relational systems have been implemented on minicomputers, but the most interesting one is INGRES. It has been developed by Stonebraker *et al.* (STON-76, HELD-75), and it has been operational since 1975 at the University of California, Berkeley, on a PDP-11 under the UNIX operating system. INGRES supports QUEL, a high level data language based on relational calculus. EQUEL is also available, which is QUEL embedded in C. Later, a casual user pictorial interface (CUPID) (MACD-75b) has been implemented on top of QUEL to provide a graphical interface to access the database. CUPID is discussed in more details in chapter 3, as it is one of the graphical interfaces which provided the background for designing PDBIS. Features which have

been integrated in the current version of INGRES, which is available as a product from Relational Technology Inc., includes user views, concurrency control, integrity control, optimizer, and recovery from soft and hard crashes.

Another area of current interest is the provision of database machines. This attempts to solve the problems of poor performance by providing specialised hardware to do some basic relational operations. Example of present database machines are: CASSM (SU-75); RAP (OZKA-77) and RARES (LIN-76). These systems are designed for cellular associative processors for performing the query, data manipulation, and data definition activities in the relational context. The design of these specialized processors consists of an array of cellular associative processors which are driven in parallel by a central controller. Each one of them is composed of a microprocessor and a segment of a rotating secondary storage device. The processing element of each associative processor performs an operation directly on its associated memory segment. CAFS (BABB-79), the ICL's Content Addressed File Store, is another system in this category. It has been developed at ICL, Stevenage, England, to support relational operations by means of a special search processor and random access bit addressable memory placed between a general purpose computer and a disc. Many other database management systems in this category have been developed after 1980 (QADA-85). One which has significant success is IDM (Intelligent Database Machine) (EPST-80), manufactured by Britton-Lee Inc. It is a back-end processor and storage system that contains a complete data management system. The machine includes a specialized hardware to perform the data management functions.

The advent of DBASE-II (ASHT-81), developed by ASHTON-TATE, made database facilities available on most Z80 based computers. This system linked with the CP/M operating system provides powerful facilities in the form of English-like commands to manipulate a small or medium size relational database. DBASE-II is

widely used for small commercial applications running on personal computers, and has been an industry standard for several years. The upgraded version of DBASE-II is DBASE-III (BIDM-84). It is a 16-bit standard database which runs with the CP/M and MS-DOS operating systems, it provides the user with a friendly front end, and it is able to handle more and bigger files than DBASE-II.

Many other data management products are available for small computers, these are either relational, relational-like, or file management packages, examples are Rescue, Delta, Superfile, Tomorrow's office (INGR-84). Other packages which are integrated with programming facilities are also available on a number of machines, such as the Lisa (WILL-83) and the Macintosh (WEBS-84), to simulate some of the office environments. These are considered further in chapter 3.

CHAPTER 2

USER INTERFACE

2.1. End-User Interface

The 'end-user interface' is that facility which enables the end-users, who are usually non-computer professionals, to access information stored in the computerized files. End users are the functional personnel in an organization who utilize data processing services to perform their work efficiently (EUFC-79). Such users are not usually proficient in data processing; their main job is something different, and data processing is just a means amongst many others they might use to solve their problems. Their use of data processing depends upon how easy those facilities are to use, their accuracy, speed, and cost.

Historically, using the computer has been limited to computer professionals, such as programmers, data processing professionals, analysts, *etc*. These people have been well trained to employ the computer to solve other peoples problems. Consequently, some of the data processing failures are due in part to the lack of understanding of the nature of the problem which is presented by non computer professionals who have a different language and logic for communication. Interfacing to the computer requires training in terminal usage and in the use of programmed and predefined parameterized formats, and skilled professionals require training in the use of time sharing systems.

Today, the end user spectrum has become much wider than before, as the computer has entered most aspects of life, and has become the direct or indirect tool for any professional. Computer based applications represent the main-line activities of enterprises, and utilizing a large database system to access the required information on the basis of day-to-day operations is becoming commonplace. Systems are now interfacing with functional end-users, administrative and professional people who are busy doing a non-computer related job. These people must access the computer

through interfaces which are easy to use and easy to learn and must not distract from their job and which must be reliable in terms of accuracy and availability.

In this chapter, the computer users are classified according to the type of their interaction with the system, and the characteristics of an end-user interface is defined.

2.2. A relational database query interface

The term 'query interface' is intended to cover those aspects of the more general term 'user interface' which are specifically concerned with querying a DBMS. The term query language was given to that self-contained high level data language which is primarily oriented towards the retrieval and the updating of data held in files or a database (SAME-81). There are several features of the query language which may be used to determine if a particular language is a query language:

(1) A query language is appropriate to be used on-line and oriented towards *ad hoc* fashion of data retrieval.

(2) The prime users of the language are those who have little or no data processing experience.

(3) The language should allow the user to specify what is wanted compactly, often in a single statement.

(4) Data entry or updates are limited to a single record or field. It is inappropriate for unlimited capabilities.

(5) The complexity of calculations and output formats is limited.

(6) A query language is appropriate when speed of development is more important than run time efficiency.

According to these specifications the query language is more applicable to data stored in the relational model than in CODASYL, or some other low level procedural oriented model.

Present-day relational database management systems provide access to data stored in the form of relations by supporting query languages which contain commands that operate directly on relations. Query languages are made available as stand-alone interfaces or through calls from programs to the RDBMS which return one data-item at a time or a set of data-items (relation) to the calling program.

Since the introduction of the relational model in 1970, various high level database languages have been introduced to support this model. Most of them had the aim of being easy to use and easy to learn by non-programmers, but the problem of the database language still needs a solution. All these languages are different, so the user may find himself having to access different databases, maybe on a different computer, with quite different languages. This problem grows day by day, as computers get cheaper and more available, and more small businesses are automating their manual data processing operations, and one can predict that computers will dominate office work, school, and home in the near future. Therefore, in order to address the non-programmer community, unified high level, flexible, user-friendly languages are becoming an important requirement (ZLOO-78). By this the user instructions in one environment will be acceptable in the other.

The current relational database query languages range from informal natural English and formal structured English to formal two dimensional and formal graphical languages. None of these languages has succeeded in being unified for a number of database systems, and none satisfy the need of the entire spectrum of potential users. It is rather the case that there are classes of languages suitable for classes of users.

2.3. User classification

Generally speaking, the simplicity of the relational data model, coupled with the relational query languages, offers the possibility of direct use of a relational database management system by non-programmers. Therefore, we can expect that RDBMS

users will differ widely in their knowledge of relations and the manipulation of relations, and also of the computer system. Thus, the potential users can be classified into a number of overlapping categories: casual users, non-programmer professionals, conventional application programmers, and database personnel.

2.3.1. Casual users

The term 'casual user of a computer system' is not a well defined term. Each specialist has his own views of what is important. It is unlikely that they all agree on the meaning of this term. Martin describes the casual user (his term is 'operator') as one who uses the terminal only occasionally, spending most of his day doing some thing different (MART-73). Such a user has little, if any, training in terminal usage. For him, "The man-machine interface must be designed to appear as natural as possible, or his bewilderment will quickly turn into annoyance, criticism, or behaviour that amounts to rejection of the system". The casual user may or may not have some programming skill; even if he has, he is unlikely to be sharply skilled by working with this system, since he uses it infrequently and has had little training with it. Codd (CODD-74) proposed the following definition: "A casual user is one whose interactions with the system are irregular in time and not motivated by his job or social role. Such a user cannot be expected to be knowledgeable about computers, programming, logic, or relations". In the first part of this statement Codd was in agreement with Martin, but the comment on motivation is surprising, as he goes on to state that "this class of users is quite broad. It include almost all of institutional management (private or public) and housewives". The definition as stated would exclude anyone who views an on-line information system as a tool to be used in his job for occasional extraction of data, but who does not need to use it constantly. Many managers, administrators, shop supervisors, and factory foremen would come into this category (CUFF-78). Perhaps Codd means that a casual user may be free to use the system, or to ignore it and obtain

the data he needs from some other source. In other words that use of such a system is not a requirement for his job.

Lough and Burns (CUFF-78), limit the term to those professionals in some field other than computing such as managers, lawyers, planners, *etc.* Generally those professionals need information which is stored in a database, but they prefer not to pay a third party (e. g. application programmer) to obtain it for them. Therefore, they chose the other alternative and interact with the system by themselves, gaining some economical benefit.

Zloof implicitly described the casual user as "The non-professional user who has a little or virtually no computer or mathematics background" (ZLOO-75), but a few years later he has accepted Codd's definition of identifying the 'true' casual user, and offered an alternative broad definition for the non-programmer professional (ZLOO-78).

As a result of a lengthy studies, Codd and others justifiably argue that "the only way to entice such a user to interact with a computerized database is to permit him free use of his native tongue" (CODD-74).

A number of ambitious attempts at supporting natural language as a database query language have been developed (THOM-69, KELL-71). These systems were designed to support one way communication (from the user to the system) in restricted English. They have not been successful in addressing the casual user problem because restricted English removes the freedom of using the native user tongue. These systems may be adequate only for those users whose involvement with the database is of such a professional or job-oriented nature and they are willing to invest the efforts and time in learning the restrictions and live with them (CODD-74).

Because of the complexity of natural language and the variations in speaking the language among people, in addition to the irregularity of the casual users interaction

with the system which implies that he might forget what he has learned in a previous interaction, an assisted instruction language is required and the formatted database should have the capability to 'talk-back' to the user in a natural language in order to clarify the user's request. To achieve this goal, Codd proposed the RENDEZVOUS system (CODD-74), which was intended to be a query formulation front end for relational database management system, in which the user types his query in natural language, and the system engages the user in a dialogue to extract information from him which is necessary to formalize queries which have been inadequately specified. Queries are only executed after the user has given his approval to the system-generated natural language statement of the query. Another natural language (NL) system is called ROBOT (HARR-78) which attempted to use the artificial intelligence (AI) natural processing techniques to translate the user request stated in English into a program that will generate the desired results. The syntactic analysis is carried out by an augmented transition network (ATN) parser. This analysis is domain independent, as it performs a search for the presence or absence of the unknown word as a string data value in each database attribute which corresponds to a possible parsing of the question. The database is therefore being used as a knowledge base system, in addition to its traditional role. This makes it simple to turn ROBOT towards understanding questions against other databases. To apply ROBOT to a new database, the database administrator must provide a special-purpose vocabulary of words and abbreviations not included in the database ATN. This process takes a considerable amount of time. One of the main objections to this system is the speed. For each question, ROBOT may search the database several times, at least once to understand it, and once to answer it.

After the problems and the requirements of the natural language have been identified, many recent attempts have been carried out to develop a natural language front end to databases independently from the applications and the underlying database management system (KAME-78), (GROS-83). Boguraev goes further in his

work (BOGU-83) by following a technique which has a clear-cut separation between the different stages of the process. The syntax and the semantic analysis are separated from the domain information in the database.

Using Natural Language for communications between people is certainly a natural way, but using it as a communication tool between user and machine is not natural unless a natural system is developed. This system should contain some sort of 'common sense', tolerate simple errors, have a large and complete vocabulary set, accept a wide range of grammatical constraints, be able to deal with partly understood input, and must be capable of providing the information and computations requested by the user (WALT-78). By providing these capabilities to NL systems, the different English spoken style can be contained, and also the operations that the current NL system cannot perform for programmers and computer professionals will be handled. Developing such a system has come to be a very important issue for the future development in computer technology. The necessity of such a system also comes from the current work in speech-recognition, which forms one of the basic activities of the fifth generation development (MOTO-84). The highest expectation for a successful NL system development is found in the research work which overlaps the artificial intelligence with database technology (ELDE-84).

In the current situation, none of the developed NL systems can satisfy the objectives of the successful database interface. Formal high level languages are still required for several reasons (ZLOO-78):

(1) A casual user does not have the knowledge of the structure of the underlying database model. Thus he can, at most, query the database, and perhaps perform a simple modification operation, but such a user cannot define relations, views, integrity and authority constraints, system catalogues, reports, *etc.*

(2) A formal high level language will always be required as an intermediate step which is used to restate the query to clarify the user's intent.

(3) Formal high level languages address a different class of users, such as professionals and professional programmers who are prepared to spend some time in learning the formal language.

(4) A high level database language can be embedded into a host classical programming language (such as PL/1, COBOL, and FORTRAN), in order to extend its capabilities.

Todays development in data processing field are more concerned about the casual user factor, and the prospective developments in office automation. In these developments one can see more graphical and icon oriented systems than linear and natural language systems. The reasons for this trend of development can be found in chapter 3 and chapter 4.

2.3.2. Non-programmer professionals

After stating the casual user class, the rest of the end user spectrum becomes more clear, especially when we draw a distinction between two types of DBMS applications, which we term operations systems, and information systems (MART-73). An operation system is one designed for the routine processing of data and the answering of prespecified set of queries, for example; a bank teller system, or a registration system for hospital admissions. Users of these systems are not professional in programming, but they are constantly using the machine unlike the casual users, and they have invested some time to learn the systems. Zloof calls this class of user a non-professional programmer and defines it as "non-programmer professional is a person motivated by his job and familiar with the particular applications. One can expect such a user to learn an easy-to-use formal language and be familiar with the concepts normally required by a relational model. In this

category one can include secretaries, clerks, engineers, analysts, *etc.*". In the second part of Zloof's definition, I agree that such a user can be expected to learn using the formal language, but I wonder why secretaries and clerks should be expected to learn anything about database concepts. In my opinion, the database concept has further application than retrieving the data, and certainly concerns another class of end users.

2.3.3. Conventional computer application programmers

In an information system, the nature of the queries will not be predefined in detail, for example, for a personal information system or a library information system. Users of these systems must be familiar with the computer system configuration, database structure and at least one computer language, and they must be willing to invest time and effort in learning complicated languages and continuing to do so.

2.3.4. Database systems personnel

They are the users who understand the behaviour of the system and any language related to it. They use the system occasionally to monitor and optimize the performance of the system. A subclass of these users usually represent a small group of people whose main responsibility is to manage the entire database system and its applications. A database administrator is an example of this subclass. A second subclass of the database personnel represents those users who built the system in the first place, and continue using it for the purpose of maintenance and optimizing the performance.

2.4. Formal high level relational language

All the current relational database languages, whether already implemented or partially implemented, come with the main objective of supporting casual users. They are of a different form, each representing the designer's view of how he can provide an easy to use and easy to learn interface to that class of users for which it was intended.

In the previous section we have established that current natural languages in which the user can use his own tongue to express his request, cannot be the form of the unified interface which can substitute for the use of the formal high-level query languages.

The state of the art of the formal high-level relational query languages can be classified into various categories such as:

(1) Relational algebra-based languages.

(2) Relational calculus-based languages.

These two classes are sub-classified into:

(a) Linear form languages;

(b) Two-dimensional form languages;

(c) Graphical form languages.

Examples of some of the implemented or partially implemented languages are:

ALPHA: (CODD-71a) Linear based on relational calculus.

RENDEVOUS: (CODD-74) Linear, Natural, based on relational calculus.

PRTV: (TODD-76) Linear, based on relational algebra.

SQL: (CHAM-76), SQUAR: (BOYC-75) Linear, based on mixture of relational calculus and algebra.

QUEL: (HELD-75) Linear, based on relational calculus.

QBE: (ZLOO-77) Linear, and two dimensional based on relational calculus.

LILA: (LINU-78) Linear, based on relational calculus.

CUPID: (MACD-75b) Graphical, based on relational calculus.

FORAL: (SENK-77) Graphical, based on binary relation model.

SALT: (KING-79) Linear, based on relational algebra.

GOING: (UDAG-82) Graphical, based on relational algebra.

DBASE-II: (ASHT-81), DBASE-III: (BIDM-84) Linear, based on relational algebra.

There is a wide range of relational systems available as operational products. As the regularity and the simple structure of the relational model are an advantages to most users, there are many other query languages offered by the manufacturers who claim that these languages are relational or relation-like, but many do not offer the operator composition of relational algebra to the interactive user, and some give no equivalent to relational join, only providing equivalents of selection and projection. Usually these systems adopt strategies to map from a complex underlying data model to the relational user's view, and this mapping may be established prior to the interactive session, and may require a considerable expert help (ATIK-81). This is certainly less satisfactory than the relational model. Most of these systems run on Z-80 or 8086 based microcomputers.

2.5. Ease-of-Use and User-Friendliness

Following the development in computer technology and the end user languages, which led to the infiltration of computer use within the non-programmer community, new terms such as user friendly, ease-of-use, human factor and psychological test, *etc.*, were added to the dictionary of computer terminology. Today the ease-of-use aspect of language is receiving great attention and has become the main objective of system design in both industry and academia, because in the end the non-programmer user must find it easy to learn and easy to use, or it will be neglected, no matter how fast, efficient, and powerful the system is (ZLOO-78).

The advertisements for most of the developed query languages say that they are easy-to-use, but according to what criteria, is still not clear. The concrete definition and measurement technique of 'easy-to-use', and how friendly the system is, is not a well defined issue. It is unlike, for example, the measurement of the performance of two machines. The terms are subjective and depend on many factors such as: user taste, background, training, and general environment.

In the existing query systems, one can notice that different approaches have been used to achieve the easy-to-use specifications. One approach is to allow the user to state his question in natural English, and the system may interrogate the user to clarify his query, such as PLANES (WALT-78). A second approach is to require the user to state his question in a formal language system, but one uses an English-like grammar and vocabulary, such as SQL (CHAM-74), IQF (BUNE-79). A third approach is to require the user to state his question in a formal language system that does not appear English-like such as QBE (ZLOO-77). A fourth approach is to avoid any typing; this is done by formulating the query through a selection of entity, attributes, and relationships, from menus or icons displayed on the screen, as in FORAL-LP (SENK-77), and LISA (WILL-83).

At present researchers and designers intend to use psychological testing as part of the design and development process (REIS-77, REIS-81). They want to use the feedback from experimentation while changes can still be made in the language without difficulty. In these experimental studies, they have attempted to define the vague notion of 'ease-of-use' and develop a quantitative measurement technique for it. This technique depends on specifying measurement primitives such as the average time required to formulate a set of queries, percentage of correct queries, confidence rating, and classification of errors.

According to one of these experimental studies (THOM-75), QBE was found to be a highly user-friendly language, and showed a wide acceptance amongst the non-programmer community. Thus there will be no surprise when the language design consideration discussed in this chapter is inspired by the design consideration of QBE.

2.6. Language design considerations

Two main factors are dependent in specifying the following design considerations: the first is the human factor represented by the user reaction toward query languages which have been under psychological studies. The second factor is due to feedback from studying the technical performance of the currently operating query language systems.

The extent to which a language satisfies the following requirements will increase the degree of user-friendliness of that language. So that these requirements should be considered in designing any database language or interface.

(1) **Minimum concepts to get the casual user started**: The system should enable the non-programmer to start with simple operations by learning very little. That can be achieved by providing the user with a teaching statement through the process of query formulation.

(2) **Minimum syntax**: The language syntax should be simple, straightforward and cover a variety of complex transactions in order not to be forgotten by the occasional users. Experiments showed that two dimensional syntax is easier than the linear syntax.

(3) **Consistency**: The language operators must be consistent through the various phrase structures. Operators must not change semantics in different contexts.

(4) **Flexibility**: The high level language must be flexible to capture the user's thought process, and provide several ways in formulating a transaction.

(5) **Procedurality**: The language must reduce the procedurality as much as possible, allowing the user to state what he wants to get, and leave the system to fulfil the request automatically.

(6) **Not sensitive**: A small change in the query (such as changing or adding a condition) should result in only small change in the query expression.

(7) **Easy to extend and modify**: The language should be able to provide the user with flexibility in reorganizing the database, by offering a dynamic modification of the database by means of creating new relations with different structure (views and snapshot).

(8) **Easy detection of errors**: The language syntax and phrase structure should be in such a way to minimize the possibility of errors; on the other hand, a simple and clear error message should be given by the system in case of an error detection.

(9) **Data Model**: The language should reflect the database structure in which the data is stored to the user in a very simple, understandable form, such as tables or forms.

(10) **Intelligent reaction**: The language should have some intelligent actions and form a sensible query for the partially specified question.

(11) **Unified language**: To minimize syntax change in adding new concepts to the language, the same syntax should be used for modification, definition, and control of the database. In a more general concept, the same language syntax and semantics should be used to access many databases represented by different data models and controlled by different DBMSs. In fact, in addition to the other characteristics of PDBIS, this is the main theme of its development.

CHAPTER 3

GRAPHICAL QUERY LANGUAGES

3.1. Introduction

Recently there has been a great attraction in using interactive computer graphics in information processing. That is possibly for a human factor, by making the user interact with the computer in an entertaining way, or possibly because it offers an alternative way to typing, and also offers the top rank of non-professional users such as managers a glorious way of getting the information they need. In fact, the graphical representation of data is more understandable and acceptable by the human brain than the textual representation. These factors and others make language designers and researchers look to use the advantage of the two dimensional nature of a display screen, to free the user from the bondage of left-to-right linear specification, and start many new techniques: such as selection from displayed menus; graphical or schematic representation of database or the query itself; several forms of feedback and result formats; having a complete picture of any part of the data structure displayed as names of relations and associated attributes on the screen; and parallel development of different parts of a query (CUFF-78, CUFF-81). Many of these features have become visible in recently developed systems, such as the STAR and the MACINTOSH-like systems.

At present there are many relational database interfaces depending on the two dimensional nature of a display screen to provide the end user with a friendly facility to access the database. These interfaces have been already implemented as products or as experimental systems. Some of them such as FORAL-LP (SENKO-76), CUPID (MACD-75a), GOING (UDAG-82), and GBRM (TESK-84) provide the facility to perform the database functions only, and so it is appropriate to call them graphical database languages. Other systems such as SDMS (HERO-80), VIEW (WILS-80),

OFFICETALK-D (ELLI-82), LISA (WILL-83), and MACINTOSH (WEBS-84), provide the facility to perform programming or information operations as well as the basic database functions; therefore it is more appropriate to call them information base systems rather than database languages. In fact these systems represent a new generation of information processing facility; most of them use the icon metaphor to represent the real world objects on the screen, allowing the user to browse through the database searching for items of interest or perform operations on these icons.

The development of the above mentioned systems and most of the recent work in this area, including PDBIS, have been influenced by three pioneering graphical database languages. These languages are Query-By-Example (QBE), FORAL-LP, and CUPID. In this chapter, I shall discuss each of them in detail in addition to other more recently developed systems, in order to enhance the freedom from the idea of linear programming specification, and to show a variety of approaches in these systems compared with the linear languages.

3.2. Query-By-Example

Query-By-Example (abbreviated as QBE) is first proposed by M. Zloof in 1974 (ZLOO-75, ZLOO-76, ZLOO-77, ZLOO-78, DATE-81), and developed at IBM Yorktown, Heights Research Laboratory. It is now running on the VM/CMS operating system. The name (QBE) refers both to the system itself and to the user language which the system supports. It is a high level non-procedural database language which provides the end user with a simple unified interface for querying, updating and control of a relational database.

The philosophy behind this language is to require the user to know very little in order to acquire the skill to make a considerably complicated query by learning the language (ZLOO-75, ZLOO-77). QBE is similar to SQL (Structured Query Language) except it was designed to use a visual display terminal, not only by displaying the

results in a form of a table on the screen, but also all queries are specified by filling tables on the screen. These tables are constructed partly by the system and partly by the user. The visible table is the framework of the language which provides the user with the contextual information of the database. The intention is that operations in the language should be similar to the way in which people use tables manually in order to find information. "The formulation of a transaction should capture the user's thought process, thereby providing freedom to formulate a transaction" (ZLOO-77).

QBE has a two dimensional syntax unlike other traditional computer languages with a linear syntax. Its syntax is simple and covers a wide variety of complex transactions; it gives the user the freedom to formulate the transaction, and allows him to create and drop tables dynamically from the database, and also provides the user with a dynamic capability to define control statement and security features (ZLOO-77).

The other major feature of the language is the use of an 'example' in the data specification, and formulating the query depends on entering an example of a possible answer in the appropriate place in an empty table. For example consider the following query:

Example QBE-1: Get suppliers numbers for suppliers in Paris.

S	S#	SNAME	STATUS	CITY
	P.S2			Paris

The query is built up as follows; the user starts by getting a blank 'skeleton' table displayed on the screen by pressing a certain function key on the terminal. Then, knowing the answer to the query can be found in relation S, the user enters S as a relation name, at the top left of the skeleton table. The system replies by displaying

the attributes within S as headings for columns in the table. Now the user might express the query by making entries in two positions (in any order). The given data 'S2' preceded by 'P.' for print which indicates the target of the query. 'S2' is a possible answer to the query which is called the 'example element', and it has to be distinguished syntactically from the 'constant element Paris' by underlining it. An example value like 'S2' is simply a place-holder; it need not appear itself in the result, or even in the original data, and the user is free to use any example value he likes. The value 'S2' is not necessary to the query and could be omitted unless it is used to established links between rows in more complicated queries.

It is clear from the above example that the user is once again required to be concerned with the relational composition of the database, something from which PDBIS frees him, where all relation names and attributes of the concerned relations are displayed on the screen. In QBE, the user must type the relation name explicitly. He has to know the logical relationship (links) between relations, (these links might not always be via attributes with identical names), and to use them to navigate through a query.

The idea of giving an example is perhaps more likely to be understood by most non-programming users than that of providing a linking variable, or that of nesting several data specifications as in SQL. However, one still has to break a question into parts which may not seem 'natural' in term of the English forms but which must correspond to the way the database is constructed from relations (CUFF-78). Then one can expect to find a simple English request which involves a complicated formulation as in the following example:

Example QBE-2: Get parts supplied by Clark.

SP	S#	P#	QTY
	S1	P.P1	
	--	--	

S	S#	SNAME	STATUS	CITY
	S1	Clark		
	--			

Example QBE-3: Get suppliers numbers for suppliers who supply both P1 and P2.

SP	S#	P#	QTY
	P.SX	P1	
	--		
	SX	P2	
	--		

In the above example SX is acting as a link between rows in the same table.

Example QBE-4: Get supplier names for suppliers who supply at least one red part.

S	S#	SNAME	STATUS	CITY
	SX	P.SN		
	--	--		

SP	S#	P#	QTY
	SX	PX	
	--	--	

P	P#	PNAME	COLOUR	WEIGHT	CITY
	PX		Red		
	--				

The above example shows that, without doubt, the navigational task becomes considerably reduced, because of the freedom to enter parts of the specification in any order. There is no need for the user, as with the navigational systems, to think about the logical construction of the query before beginning. He can enter his query in any order he likes; the order of rows within a query table is totally immaterial; and the order in which the user fills in all the entries constituting these rows is also arbitrary. For instance in the above example, one can treat the query as follows: 'Get red parts,

then get the number of supplier supplying these parts, and then get the corresponding names.' In such case the user probably completes the query in the order P, SP, S or he may think about it in a different way. Whatever is the way, the final query is the same. In other words, QBE is a nonprocedural language capable of supporting many different ways of thinking about the same problem. Questions which require careful thought in order to find a strategy for solution in some other languages can be put together (usually) in a straightforward manner.

Example QBE-5: For each supplier get part number and names of all cities supplying the part.

The result of this query will be a projection of a join of two tables. The user first has to create a skeleton table the same shape as the expected result (with the appropriate number of columns). This table may be given any names or it may be left unnamed. The user can use the result table with the existing two tables, to express his query as follows:

S	S#	SNAME	STATUS	CITY
	SX			SC
	--			--

SP	S#	P#	QTY
	SX	PX	
	--	--	

RESULT	P#	CITY
	P.PX	P.SC
	--	--

As we notice in this example, there are three links; SX between tables S and SP, PX between tables SP and RESULT, and SC between tables S and RESULT.

Example QBE-6: Get all pairs of supplier numbers such that the two suppliers are located in the same city.

S	S#	SNAME	STATUS	CITY
	S1			CX
	--			--
	S2			CX
	--			--

RESULT	FIRST	SECOND
P.	S1	S2
	--	--

CONDITION
S1 < S2
-- --

In some queries it is not possible to express conditions within the framework of the query tables. In such cases QBE has another simple and powerful device, the 'condition box', which can be obtained by using another function key on the terminal, and used to contain the condition as in the above example.

The examples so far shows that AND or OR operations can be expressed entirely implicitly, but it can be made explicit and perhaps reduce the amount of typing as in the following examples:

Example QBE-7: Get supplier numbers for suppliers who supply both part P1 and part P2.

a) Implicit AND

SP	S#	P#	QTY
	P.SX	P1	
	--		
	SX	P2	
	--		

b) Explicit AND

SP	S#	P#	QTY
	P.SX	PX	
	--		

CONDITION
PX = P1&P2

A number of built-in functions with the 'ALL' operator are provided by QBE, CNT.ALL, SUM.ALL, AVG.ALL, MIN.ALL, MAX.ALL. 'ALL' is always specified, and

'UNQ.' is the optional operator, specified to eliminate redundant duplicates before applying the functions (CNT., SUM., and AVG. only).

Example QBE-8: Get the total number of suppliers.

S	S#	SNAME	STATUS	CITY
	P.CNT.ALL.SX --			

Example QBE-9: For each part supplied, get the part number and the total quantity supplied of that part.

SP	S#	P#	QTY
	P.G.PX --		P.SUM.ALL.QX --

The 'G' is the QBE equivalent of the SQL GROUP-BY operator. In Zloof's papers the grouping is accomplished by double underlining for an explicit group by, so, P.G.PX is equivalent to P.$\underline{\underline{PX}}$.

Query-By-Example can also access data stored in a hierarchical data model (ZLOO-76). Recently QBE has been extended to involve word processing, mail systems, and other facilities required in office automation (ZLOO-82).

3.2.1. Experimental study

The main approach to evaluate a proposed query system is to find how far the system is easy to use. One approach of an easy to use system is to allow users to express questions in natural language 'English'. A second approach is to require the user to state his question in a formal language system, which has an English like grammar and vocabulary, such as SQL. A third approach is to require the user to state his query in a formal language system that does not appear to be 'English-Like'. QBE

is best described by this approach (THOM-75).

It may seem that the approach which allows the user to state his query in natural English must be the best approach toward an easy-to-use and easy-to-learn system. The truth is not so; many people have a considerable interest in QBE and its promising future for non-programmers to learn and use, even before it was implemented. The first reason for this is that it avoids the restrictions on vocabulary and syntax which might exist in natural language that the user has to keep in mind. The second reason is that, as the user becomes more familiar with a system, feedback dialogue may come to be perceived as a waste of time, Thirdly, this dialogue will increase the cost of the computer system. A fourth reason relates to data representations; related studies show that data represented in natural language is not optimal for humans (THOM-75). Providing the user with a formal representation may help him better to formulate and solve the problem.

Thomas and Gould carried out experiments on the usability of QBE before it was implemented (THOM-75). 39 student and recent graduates, (only four of them had some minor programming experience) were given three hours of training, with two 20-question tests in the middle and at the end. Training consisted mostly of about a hundred examples, rather than detailed explanation. During testing examinees were not given feedback about the correctness of their answers.

67% of the test questions were answered correctly, a result which is slightly better than in Reisner's SQL experiment which involved 14 hours tuition. The range of individual's differences was 0.33 - 0.93 on the proportion of queries correct, so it is difficult to generalise here about QBE suitability for computer-naive users.

Thomas and Gould comment that the proportion of answers correct would probably have been higher had a live QBE system been available, as the repetition of the same syntactic mistake in several different queries would be reduced. But one can

see in the implemented system, practical query formation involves the use of several keys for creating new table skeletons, widening a column to take a long entry, defining a condition box, *etc*. In addition, the paper test showed a representation of the database already broken down into relations with their attributes; in a real system, a user would have to discover the structure of the database and select the appropriate relations. These factors tend to increase the user errors in a live system. This experiment showed two major type of errors: confusion between built-in functions, especially ones for counting and summing; and difficulty in specifying the universal quantifier when it was required.

Another experiment had been carried out by Greenblatt and Waxman (GREE-78), comparing the performance of a group of students who were taught QBE, SQL and relational algebra. This experiment succeeded in giving better comparisons than Thomas and Gould, who tested QBE only and compared their result with Reisner's experiment on SQL (REIS-77), which was obtained under different conditions. This experiment was also carried out as a paper test, and showed 75.2% queries were correct for QBE; 72.8% for SQL; and 67.7% for algebra. Even though the differences are not very significant, the mean time per query was much less for QBE (0.9 minute) than for other languages (2.5 and 3 minutes respectively). This gives the user quicker feedback on errors, and helps him to reach a successful query faster, and gives the credit to QBE.

3.2.2. Summary

There is no doubt now about the easy use of QBE compared to most other query languages. The user needs to know very few basic QBE concepts before being able to pose useful queries, so that people motivation toward the language is high. According to the Thomas and Gould experiment users required about one-third the training time, were about twice as accurate in writing queries and some were faster than the other competitive query languages like SQL and SQUAR (THOM-75). It is a great advantage

to be able to enter part of the query in any order, and it is also a good idea to have alternative ways of formulating certain queries. Logical manipulation is much reduced by having implicit AND and OR operators.

On the other hand QBE is still a navigational language, and the user needs to know relation names and the semantics of the links between them. The system requires the exact specification before it will retrieve any data. The user has no reassurance that his query is correct, because there is no feedback other than the output result. The physical problems of manipulating skeletons, columns and cursors may reduce the elegance of the system.

3.3. FORAL-LP

FORAL-LP (FORAL with Light Pen) (SENK-76, SENK-77), developed by Senko in 1976, makes more effort to avoid typing than other graphical data languages. It was an experimental system which combined menu-selection with a displayed network representing a database. It used the light-pen as a communication device instead of the keyboard. Using this, the user can specify database attributes or combination of attributes, connection paths between them, and any logical operations on them, simply by pointing at the appropriate area of the screen. Even constants can be specified using the light-pen, since the alphabet and digits are permanently displayed across the top of the screen, and a constant can be constructed by repetitive selection from this set. Therefore the user can avoid any typing.

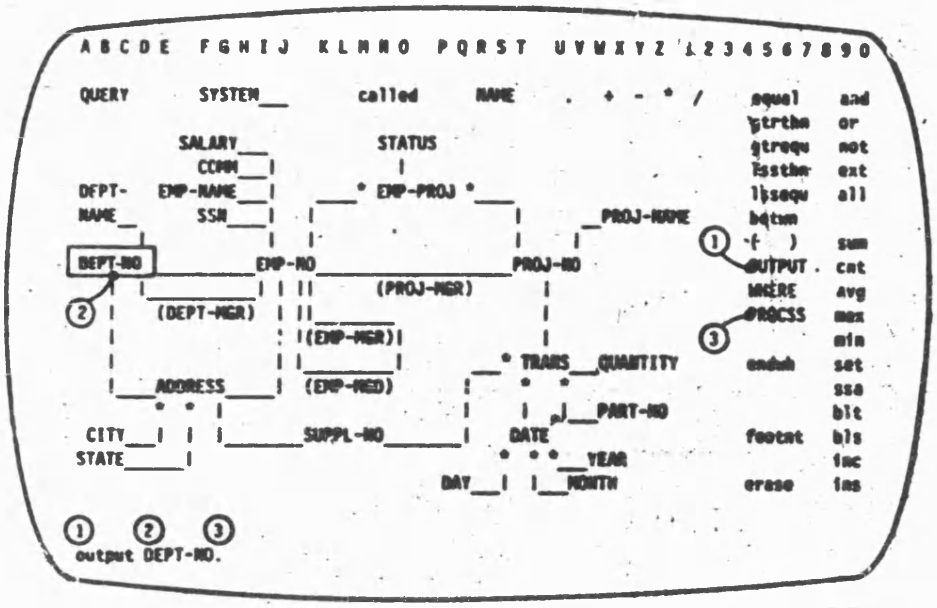


Figure 3.1 Screen layout.

As shown in Figure 3.1, the screen is divided into three distinct areas. The bulk of it is devoted to a network representation of the database. To the right of this area, there is a two-column menu for logical comparatives and connectives, built-in functions, instructions or delimiters for the language interpreter. There is another line of four items plus some arithmetic symbols below the alphanumeric row. The third area of the screen is a line at the bottom in which a linear language representation of the query will appear. In the network of Figure 3.1, each node (in capital letters) stands for set of names or values. For example, 'EMP-NO' stand for a set of employee numbers. Each name in the set stand for a real world entity. For example the number 100, in the set EMP-NO represents the real world EMPLOYEE with number '100'. Each arc stands for a type of association between two real world entities. Most arcs do not have names on them in order to allow the same name to be used in two different contexts, which stand for two different things, so the user can operate with a smaller set of names. At each position in the structure of a FORAL statement, the FORAL

context points to one and only one of the nodes. Arcs connected to this context can be addressed uniquely by using the name of the node at the other end of the arc. For example, in Figure 3.2, when the context is at ADDRESS, then the line leading from ADDRESS to EMP-NO will give values for the attribute 'employee of address'. Similarly the line leading to DEPT-NO will give values for the attribute 'department of ADDRESS'.

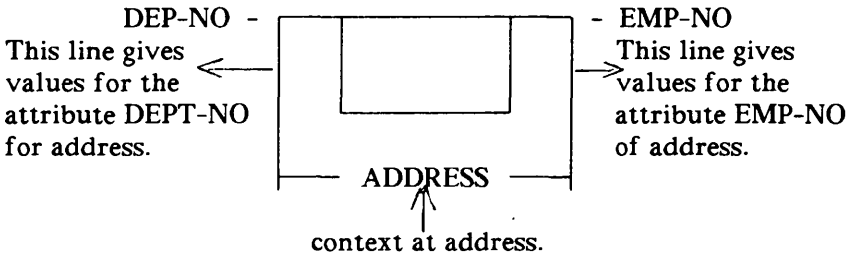


Figure 3.2

We also notice in Figure 3.1 that some of the nodes represent the database attributes, while the others (such as EMP-PROJ) are essentially a place-holder, representing the logical combination of those nodes (such as PROJ-NO and EMP-NO) connected to it by starred links. In Figure 3.1, the light-pen path is indicated by numbered circles. The numbered circles above the bottom query line indicate the portion of the query statement constructed by the corresponding pen touches.

In FORAL-LP, the entities and facts in the user's application area are represented by the binary semantics net. The user writes programs by making his way around this net with a light-pen. The system translates the light-pen touches into a FORAL-II statement for subsequent compilation. Both the FORAL statement and a picture of the light pen touches are displayed to provide feedback to the user.

Each FORAL statement is constructed in two steps:

Step 1: The user defines the kind of entity name and attribute fields that are to appear

in the output (For example, he might define two fields, one to contain addresses and the second to contain departments at each address).

Step 2: The user places conditional limitations on the content of the fields.

Experience with FORAL-LP showed that this two step procedure is not appreciated by the users, because specifying actions by more steps, generates more opportunities for errors.

The user always starts his query by touching 'OUTPUT' ((1)) on the operation menu (Figure 3.1). He then touches the name of the entity set that is to be the subject of his report, 'DEP-NO' ((2)), this pen touch establishes the 'context' at the 'DEP-NO' node. This node will be intensified on the screen and a linear FORAL-II statement will appear at the bottom of the screen. The user always completes his query by touching PROCSS, where FORAL-II print out a list of 'DEP-NO's' as in Figure 3.3.

FORAL-II statement	Printout:
OUTPUT:	*
DEP-NO	DEP-NO
	400
	300
	450

Figure 3.3

The asterisk indicates that the context is located at 'DEP-NO'.

If the user wishes to limit the values that will appear in some of the printout field, then he can go to step 2, and touch 'WHERE' in the operations menu instead of touching 'PROCSS'. At this point the query diagram will be presented. The user will then touch the attributes or entity set name that he wants to become his target. The system will then display a diagram of the network with only the target node intensified. The user can then proceed to define test as shown in Figure 3.4. He ends

the definition of the test for a particular target by touching 'endwh'. The user now can select another target by touching 'WHERE' again or he can touch 'PROCSS' to end his query.

Example FORAL-1: Print departments and for their employees whose EMP-NO is over 432, print name and salary.

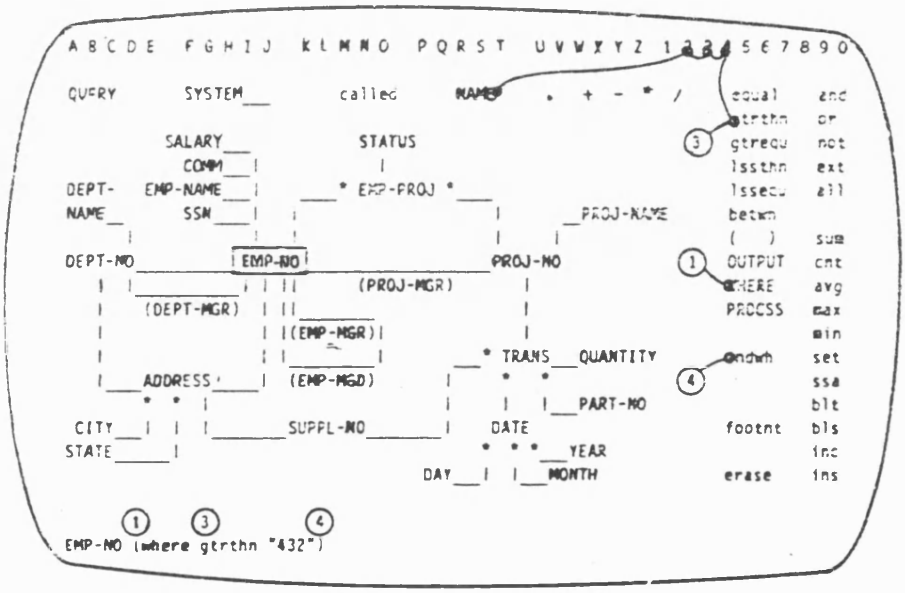


Figure 3.4

In the output step (step 1), the user define the following query which will appear when he touch 'WHERE'.

output
DEP-NO ②
for EMP-NO
print
NAME,
SALARY.

After the user indicates the 'target' by touching 'EMP-NO' on the query diagram, the

network will return with 'EMP-NO' intensified, as shown in Figure 3.4. The user can then define the test condition on the values by touching 'gtrthn', *etc.* The system will insert the test condition in parentheses immediately after the name of the target. We notice that touch '2' is on the query diagram, so it is not on the network screen.

3.3.1. Summary

In this summary the positive and the negative aspects of this interface are presented. On the positive side the interface succeeded in avoiding typing or even having a keyboard. But lately, it has been found by experimenting in PDBIS that typing a constant value is much easier, and needs less time than selecting each character of it from an alphabetical set.

Although the user must navigate long connections between attributes, the method is still easier than with linear languages. The user does not need to know the data structure; the network on the screen shows the logical relationship between entities, so the physical representation of the database is hidden from the user, and it may be changed without effecting him. In other words there is a high data independence in this interface.

The FORAL-LP users can think in terms of real world entities and associations rather than records and files represented in the computer. light pen entry removes the need for typing skill, and speeds up transaction entry. It also removes most possibilities for spelling errors. FORAL-LP takes advantage of the facilities of a screen in several ways. Selection is made through a single movement with a light-pen instead of by explicit typing; it uses schematic graphical display and extensive menus; and the user is provided with immediate graphical feedback on his transaction in two forms: firstly, each element of a chosen path is highlighted immediately after it is selected, so the user can review the query before sending it to be processed; secondly FORAL-II statements being created by the system provide another check to the user. Possibly

this check is not very helpful to the casual user, who does not know the context of FORAL-II, but it should enable him to check at least the spelling of constants and their association with attributes in the database.

The menu technique provides standard functions, and could be extended to application-dependent functions, provided the underlying FORAL-II system can handle them.

The other aspects of the language, which can be considered as the weak side, can be summarized and discussed as follow:

If we look at the examples in Senko's paper (SENK-77) we find that the query formulation process is really complicated and shows that the user will get lost before he finishes his query construction, and that appears to be not only for the casual user but also for the regular trained user. One feels that a lot of practice and acquired skill would be needed to answer other than very simple queries. Probably the situation is all right when there is a simple clearly logical structure, but the case is totally different when there is a large complicated structure; it looks cluttered and confusing on the screen. Senko points out in his papers that there are techniques which could help here, such as 'windowing' the screen so as to look at only one part of the database at any one time. In fact, these make extra difficulties for the casual users and detract from the advantage of being able to scan the database's logical structure easily. The system would also lose the property of the feedback through the highlighting of the navigation path, and put extra work on the user to memorize what he had done. It is a pity that the user still has to navigate between entities and their relationship.

The user has to be very careful about whether to select a node or a link from the network when he builds a path. These selections need some understanding of the FORAL principle of context. Some of these decisions may be puzzling and make path selection a hazardous work.

Screen size is another limitation to the interface especially when there is a large data structure to be displayed. The light-pen is not the best tool to be used. For many users it is an unfamiliar tool to manipulate, and requires fairly precise use to select the correct part of the screen.

Having two steps to form a single query is not preferable to most users as Senko noticed himself. The user would be more successful with a system which reduces the number of different actions to think about at any one time (CUFF-78).

One of the problems which might occur in building the query is that if in the definition of an indirect attribute the path goes through the context more than once, then there will be an ambiguity and confusion in the meaning of the pen touches. The way of resolving this ambiguity by touching 'called' on every touch to the context except the last one, is quite boring to the user. And also by going through the same path more than once, the feedback which represented by this path will lose its meaning to the user.

FORAL-II feedback may be useful to a person with some programming experience, it is not suitable for other classes of users. Since FORAL-II is a language for a computer, there is no reason to be displayed to the user. It would be better to be converted to English-like style and displayed as feedback.

3.4. CUPID

CUPID (Casual User Pictorial Interface Design) (MCDO-75a, MCDO-75b) is a facility designed to support casual users interactions with a relational database system. It is the front end user interface for the relational database system INGRES which has been implemented as an experimental system on a PDP-11/40 at the University of California, Berkeley. Like FORAL-LP, it uses schematic diagrams; menus and a light-pen, but here the query is diagrammed by the user with 'menu-move' operations, rather than displaying the database network on the screen. It compiles the query

diagram that the user creates into 'QUEL', which is then passed to the underlying INGRES relational database management system.

In this interface the screen is laid out in three distinct areas as shown in Figure 3.5. The first area contains instructions on how to proceed, the second area contains a menu of CUPID commands, and the third area is the working area which is further divided into three parts as shown in Figure 3.6.

.....INSTRUCTIONS	
WORKING AREA	COMMAND MENU

Figure 3.5 CUPID screen layout.

QUERY DIAGRAM	
MENU OF PREVIOUSLY SELECTED NAMES OF RELATIONS & ATTRIBUTES	MENU OF PICTURE SYMBOLS

Figure 3.6 Working Area.

The user build his query by combining two functions:

(1) Using the light-pen to select from the menu of relation and attribute names.

(2) Query drawing-selecting from the menus of command names and picture symbols.

Each selection may require further selection of points on the screen in precise order.

Example CUPID-1: List the shipment relation.

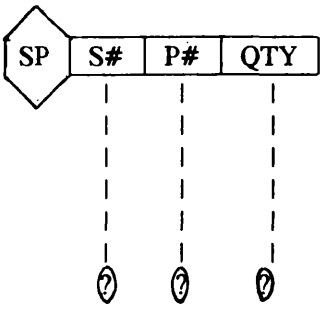


Figure 3.7

The query could be created by the following steps:

(1) Select the hexagon shape (to contain the relation name) from the symbol menu, and place it on the modelling space. (this is done by two light-pen hits.)

(2) Select and place three rectangles (attribute-names holder) as in Figure 3.7 (this step take 6 light-pen hits).

(3) Select relation 'SP' and its associated attributes from the name-menu and place them in hexagon or rectangle (it takes 8 hits).

(4) Select and place three upright oval containing '?' (6 hits).

(5) Select the connector command each time, connect each attribute box with a '?' oval (9 hits).

(6) Select the finish command (1 hit).

We notice that the special character '?' indicates what the user wishes to see (target list); the ordering of domains is immaterial; the sequence of steps in formulating the query is immaterial. This is one aspect of similarity between the language and 'QBE'.

Example CUPID-2: Get supplier names for suppliers who supply at least one red part.

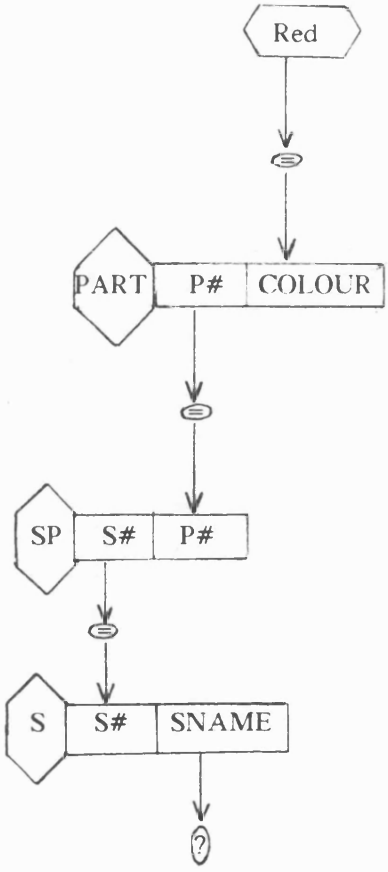


Figure 3.8

Example CUPID-3: Get supplier number for suppliers with status value less than average status.

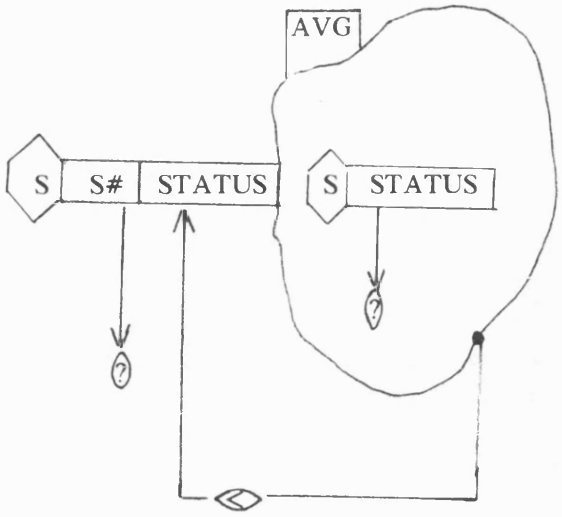


Figure 3.9

In a simple query like the one in Figure 3.7, it take 32 light-pen hits to be formed, what about a little more complicated query? This is really a large number, especially if we consider the user is continually moving between menu and query-area on the screen; moreover, the constants such as 'Red' in Figure 3.8 have to be typed between light-pen movement. Switching attention between two input media is unappreciated by the casual user (MART-73). FORAL-LP successfully avoided this trap. CUPID appears to require more patience, accuracy, and attention than the casual user may have.

In this interface there is no feedback which could allow the user to check the system's understanding of the query. The query picture is not always adequate for this.

In some queries the same pictures have to be created more than once (Figure 3.10), it is waste of time; effort, and memory space, beside the ambiguity and confusion it causes.

Example CUPID-4: Find all parts whose quantity supplied are greater than the

quantity supplied by P2.

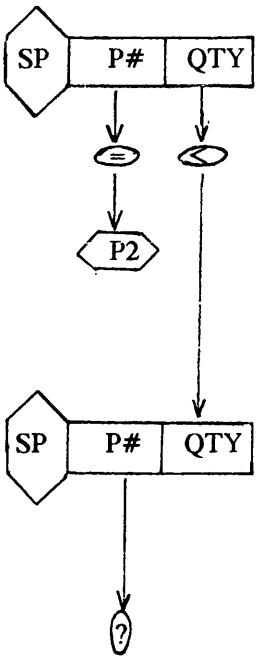


Figure 3.10

3.4.1. Experimental study

McDonald ran an experiment to compare the usability of CUPID and QUEL (MCDO-75a). 22 people were tested, 19 of them with no computer experience. This experiment was carried out on a working system. After having been taught each language, the participants were set questions most of which related to the actual relation and attribute names, and some of which advised on the joins to be made between relations.

Since CUPID is based on top of QUEL, it is certainly does not have the direct implementation of the relational calculus, and it is not surprising that more CUPID than QUEL answers were correct in this test. There were 70% as many errors with CUPID as they were with QUEL. 64% of the errors with CUPID were due to misconnected symbols or problems with linking relations. These type of errors arise

because the user has to do the navigations through the database.

3.4.2. Summary

Generally 'CUPID' does not appear to be very suitable for casual users, even though 79% of the inexperienced subjects preferred it to QUEL (CUFF-78). Perhaps the pleasure it appears to give when first used can be the motivational factor for casual users, but the amount of training, care and effort required to work effectively with it may stand against it.

3.5. Conclusion

No doubt these systems have many different superior approaches toward an easy to use and easy to learn interface than those in the linear specification procedural languages. And certainly these approaches will have their influence on any future graphical end user database interfaces.

QBE seems to be more complete and more realistic in representing the data structure, and also it seems more solid to be used by a wide spectrum of end users. But it is still far form being an optimal end user language.

Neither FORAL-LP or CUPID look as though they would be particularly convenient to use; they appear rather clumsy and long-winded, and normal sized screens could certainly look cluttered if used with either of them. However, one can assume that better engineered and more convenient interfaces based on similar principles will be developed. Their positive aspects will be reflected in the next wave of the end-user systems: schematic layout, graphics, visual prompting, pointing, various kinds of feedback, no explicit naming of database components, freedom of choice in the order in which input is entered, colour too, will be a useful addition (CUFF-81). Several of these features have been shown in recent developed systems, as they are explained in the following section.

3.6. Spatial and Iconic Systems

The principal advances in graphics over the last ten years have been economic; inexpensive raster display have made good-quality interactive graphics affordable in computer terminals, and together with high performance microprocessors have led to the 'workstation' with an integral display. There have also been gains in graphics hardware, such as improved colour displays, raster printers, and inexpensive input devices such as the Puck (Mouse). These advances combined with the need for an easy to use interface to be part of office automation development, have led to the development of a number of systems in both academia and industry. These systems depend heavily on graphics to provide a better way of interaction between the user and the data. In these systems the real world data objects, especially the desktop, are represented as icons on the screen, so that actions on the data objects can be performed by making actions on these icons using an efficient pointing device like the Puck.

Although these systems are connected with the general information system, and office automation, rather than representing interfaces to particular DBMSs, they fall in the same category as PDBIS in adopting the same technique of selecting the real world objects and in using the same mapping method. These systems will have their influence on the future improvement and development of PDBIS. Therefore some of these systems will be discussed in this section to show the techniques which have been used in these systems to represent the office environment.

One of the first systems in this trend is the Spatial Data Management System (SDMS) (DONE-78). It is a prototype system developed at MIT in 1977. The idea behind this system is to emphasise and encourage user ability to manage information spatially, and use spatial clues to locate and access the data. The real-world represented in the database in a form of alphanumeric, graphic, photographic, and film data. Information is represented by a series of pictures (icons), presented to the user

on a large rear-projected colour television screen. These icons usually corresponding to entities in the database.

A system called VIEW (WILS-80) developed by Computer Corporation of America (CCA), built upon the same idea and technique of SMDS, but it is more realistic and less expensive.

Zloof has extended the concepts of QBE from the structured database environment into the office world by developing Office-By-Example (OBE) (ZLOO-82). This system is an attempt to combine and unify aspects of word processing (editing, formatting), data processing, report writing, graphics, and electronic mail.

For several years, a major research effort at the University of Toronto has focused on developing a distributed office information system built upon a database management system using forms as the visible system object. This system is called Office Form System (OFS) (TSIC-82).

A prototype office information system has been designed and implemented at the Xerox Palo Alto Research Center called Officetalk-D (ELLI-82). It is concentrated upon the aspect of distributed control and distributed data manipulation in office automation systems. The first version of Officetalk system was Officetalk-Zero which focused upon understanding the issue of interfacing between users and machines. It provide a good individual user interface, but did little to unify forms within an office or give the manager a clear picture of the workflows in the office. The desktop of the user interface in this system consist of;

(a) In-Basket- an index of incoming mail;
(b) Out-Basket- an index of mail to be send, and mail that has recently sent;
(c) Form Index- set of forms saved by the users;
(d) Blank stock Index- set of available forms.

Actually this system presents a more sophisticated user interface than OBE and OFS, but the underlying data manager lacks the power of the other systems.

In 1981 Xerox announced the 8010 Star Information System (SMIT-82a, SMIT-82b). It is a new personal computer designed for office professionals who handle information. It is a multifunction system combining document creation, data processing and electronic filing, mailing, and printing. The system emphasis on graphics to create an electronic metaphor of a physical office. Data processing deals with a homogeneous databases that can be sorted, formatted and filtered under user control. In this system the office environment is represented on the screen in the form of icons, so that one can see document, folder, file drawers, in-basket, and out-basket, or even a telephone as a small picture or object on the screen. These objects can be selected by using the Mouse device.

Similar ideas of using icons to represent the desk top and other office physical objects on the screen, can be found in Lisa (WILL-83) and Macintosh (WEBS-84). These systems are developed by Apple in the recent years to provide an integrated office system for the single user. Lisa and Macintosh are personal computers depending on the graphics heavily to offer the user a graphical interface to use the system and provide a sort of office information system. The desk top on the screen hold icons representing clock, calculator, clip board, trash-can and two special items called 'ProFile' and 'Preferences'. Selection from the screen is also performed by using the Mouse device.

Many other systems like Smalltalk-80 (GOLD-83, XERO-81, TESL-81), Message Management System (LEE-83), Text Editor System (JONG-82), have been developed to facilitate the interaction of humans with the machine. These systems will certainly support the office automation directly or indirectly.

However, PDBIS should be a further step toward the optimal end user database interface, and open new approaches to achieve its objectives by employing the recent development in computer graphics and the attached hardware.

CHAPTER 4

PDBIS: Perq Database Interface System

4.1. Introduction

As it has been concluded in the previous chapter, none of the existing database interfaces have succeeded in providing easy to use facilities and satisfying the requirements of the entire spectrum of end-users. In this research, PDBIS is a graphical end-user interface which has been designed to provide the end-user with a unified, easy to use facility to access a number of database systems controlled by different database management systems. Thus the user of PDBIS has to be familiar with only one unified interface instead of numerous interfaces to utilize different DBMSs.

In the design of PDBIS many of the weak points which have been identified in the comparable graphical interfaces are avoided, and many superior properties are added in order to achieve the primary objectives of the interface. These properties will be elaborated through this thesis.

The use of the graphics techniques in PDBIS as an alternative to natural language to provide the unification and easy-to-use specification is supported by two main factors; the first comes from the feedback of the traditional graphical database interfaces (the interfaces of the 70's, such as CUPID, FORAL-LP, and QBE), while the second factor comes from the recent development in computer hardware and its attendant powerful graphics software, which is sophisticated enough to provide many functions required for an easy to use interface. These facilities have to be developed by the interface designer himself in the case of adopting the old graphics techniques or using other techniques rather than graphics. This advancement of graphics technology and its applications in data processing is reflected in the development of the current wave of systems like LISA (WALL-83), STAR (SMIT-82b), MACINTOSH (WEBS-84), and others. Other motivations which led to the use of the graphics techniques in

developing PDBIS are summarized by the following:

(1) Experimental studies on graphical interfaces have shown that the user is more successful in formulating his query in a graphical language. A great deal of information can often be shown more precisely, accurately and clearly in a two dimensional syntax rather than in one.

(2) Restricting the representation to a specific graphical form instead of allowing any number of words and phrases, will help in making the graphical language a finite state grammar. This will eliminate many of the anomalies and complexities of the non-finite state grammar, English.

(3) Fewer typing errors will occur because only a very small portion of any query expressed by graphical representation need be entered at the keyboard.

(4) Procedural and ordered manner of phrasing will be a very much reduced. This is a very important matter in achieving the easy to use property. Generally, procedural and ordered expressions are found in linear format specifications.

(5) The graphics is also easy to use in other language (French, Japanese, Arabic, *etc.*) rather than English (Natural language).

(6) Given the preceding five points and the absence of the other points of difficulties existing in English language based implementation, one can expect that the implementation effort is much reduced over the effort necessary to implement a natural language interface.

4.2. System overview

PDBIS is a front-end user facility designed to bring the casual user as well as programmers into effective communication with formatted databases. It is another interface which lies in the family of the two dimensional language, which is intended to take advantage of the two dimensional nature of a display screen and free the user

from typing. It offers him the capability to select from menus, employing the capabilities of the Perq display screen and its graphics package to a great extent, and capitalizes on the software cursor with the movement of Puck interactions to formulate a request to a database.

PDBIS was implemented on the Perq machine which acts as an intelligent terminal to manage the interface and connect the end-user with the host machine where the database management system and the data reside.

The user expresses his query in terms of selected real world entities (relations and attributes), comparison predicates, and functions. The selection of these primitives depends almost entirely on menu-selection operations on the Perq screen. By moving the Puck on the tablet which is followed by the movement of the software cursor on the screen, a particular spot can be specified to be selected. This spot is indicated by the cursor position on the screen. Then, the selection operation is accomplished by clicking the yellow button of the Puck. This method of selection is very easy and accurate. The user can be very precise in his pointing to any spot on the screen, aided by a software cursor which can be formatted to be suitable for pointing in PDBIS layout. The tool of the selection does not need any previous familiarity, and does not have any complication in its use, as it has been used in many recent iconic systems such as the Macintosh and the Amiga, unlike the use of light-pen in FORAL-LP and CUPID.

PDBIS has been designed to be implemented in two different environments of database processing. In the first environment, PDBIS offers the user a unified facility to access databases controlled by different database management systems which are usually operating in different computers. Thus, the power of PDBIS for supporting retrieving and updating in this implementation is actually reliant on the power of the underlying DBMS and its access language. In the second implementation, PDBIS provides a unified end-user facility to access a heterogeneous, distributed database

system. In this implementation the user formulates his query on the Perq or on a similar machine, while the data is accessed from a number of databases operating in a number of machines connected by a communication network. One of the database management systems in the network acts as a central management system, which will handle the final execution of the query. In this implementation the access power of PDBIS is reflected by the power of the central DBMS. One of the main objectives of this implementation is to investigate the feasibility of using PDBIS in office automation.

For the purpose of this research PDBIS has already been implemented on top of four different database management systems; system-R (ASTR-76), LINUS (LINU-78), CODD (KING-83), and DBASE-II (ASHT-81). In other words four different analyzers have been written to translate the query from PDBIS representation to SQL (CHAM-74), LILA, SALT (KING-79), and DBASE-II commands, respectively. In addition, in the early stages of the implementation, a simple file management system which is able to perform selection, projection, and join operation was developed on the Perq to provide a preliminary test bed.

Implementing PDBIS on different database management systems, has given a better chance for evaluation of the interface, as well as providing feedback from different environments, which leads to faster improvement and widens the generality of the interface.

The syntax of PDBIS covers almost the entire set of functions, operators, and the operations which exist in the SQL database sublanguage. SQL is one of the most comprehensive and complete relational database languages, which has been designed to provide all user requirements. This language supports all levels of data processing such as retrieval, manipulation, definition, and data control. By having a SQL-like system working underneath the interface, PDBIS inherits its capabilities and gains the generality which enable PDBIS to cover any other database query language syntax

which constitutes a subset of SQL syntax.

LINUS is another powerful, inquiry and update system for accessing centralized Multics Relational Database Store (MRDBS). LILA, its interactive data sublanguage, provides retrieval and update facilities. The reasons for choosing LINUS in the current implementation is to provide experimentation in accessing a database system operating in a large scale computer using PDBIS from minicomputers like the Perq. This implementation will also offer the large number of Multics users the chance to utilize LINUS through an easy to use, friendly interface.

The implementation of PDBIS on top of CODD provides the feasibility study for using PDBIS to interface with a special purpose relational database management system operating in a minicomputer such as the DARKSTAR-68000. CODD has been designed to manipulate a large collection of historical data, through its interactive data sublanguage SALT. CODD currently operates on a DARKSTAR-68000 minicomputer at the University of Bath, and it may be chosen to be the central DBMS in the future implementation of PDBIS-distributed database system, details of which are given in chapter 6.

The other implementation of PDBIS shows the usability of PDBIS with the most widely used microcomputer,' the commercial DBMS 'DBASE-II', which operate on microcomputers based on the Z80 microprocessor.

Currently both CODD and DBASE-II are providing a live implementation for PDBIS. LINUS is already operating on the University Multics which can be easily connected to the Perq and utilize PDBIS.

One important property which distinguishes PDBIS from the interfaces which have been presented in chapter 3 is that PDBIS provides the user with more than one method of feedback. The colour of any selected element on the screen is inverted, so that at the end the elements in black represent the user's query on the screen. The

element which might be selected more than once cannot be distinguished by its colour, because the colour will be inverted twice. Therefore, another style of feedback is used to indicate the selection of such elements to the user: all the selected elements and any typed-in constant values are displayed in the selection order, in a small window on the bottom left of the screen. Elements in this window can be scrolled up and down to become visible to the user. A further method of feedback is represented by the query itself when it is presented back to the user in the formal representation of the specified database language. This feedback might be more useful to a user who is familiar with the underlying database language. Unfortunately correcting any mistake during the formulation process of the query is impossible, the user has to restart again constructing his query; but this is not a difficult process, due to the easy and fast method of pointing. The error messages for any invalid query will be displayed to the user in the main window on the screen. These messages are in fact initiated by the underlying DBMS rather than by the PDBIS; therefore they may not be understood by the naive user who is not familiar with the underlying system. However PDBIS provides an easy, efficient, and accurate query formulation method to the naive user who usually issues simple and clear queries rather than complicated ones. This certainly reduce the error rate, and providing an error messages interpreter to PDBIS becomes unnecessary.

The formal representation of the query and the output result are saved in two external files until a new query is issued and new result is accessed. This enables the user to execute the query more than once, simply by selecting the ENTER command, and also allows resulting values to be used for purposes other than viewing. Prompts for instructions are displayed on the top of the screen. This single statement is changed as the user proceeds in the formulation of the query. It prompts the user, giving instructions on how to start his query and how to proceed. A complete output facility for formatting the output result and for report generation is connected to PDBIS.

4.3. Interface objectives

Before designing the interface the following primary goals were set:

(1) Simplicity.

(2) Ease of use.

(3) Minimum training.

(4) Portability across different DBMSs.

(5) Cover most of the facilities of the underlying database languages.

(6) Suitability for use by all levels of user.

In addition to the above primary objectives, several properties have became apparent during the implementation of the interface;

(1) Friendliness to the casual users. By using the graphics facilities and different kinds of feedback, the interface provides the user with a pleasant environment in which to work.

(2) It provides another way of employing the interactive graphics technique to facilitate the access of the database; other than the ways which have been used in CUPID, FORAL-LP, and QBE.

(3) It provides the users with a new look to the database structure. The database is presented to the user in a very simple, natural form. The user is entirely separated from the logical as well as the physical representation of the database. He is not required to remember any relation or attribute names or their relationship. They can all be displayed to him on the screen under his control.

(4) The user of PDBIS is entirely separated from the database language and its complications. He is not concerned with how the data will be accessed, or about the formal representation of the query, or the DBMS. This is the main property of the unified interface. PDBIS is not only suitable to be used by the entire

spectrum of end-users, but also provides the same facilities to access databases controlled by different DBMSs.

(5) Extensibility: the generality of the interface, and the separation of the user and the interface from the underlying DBMS and the database language, makes it very easy to include other DBMS to operate underneath the interface. This is done simply by adding its analyzer to the system.

(6) It provides interactive graphical methods for accessing a large database which may reside on a large scale computer system from a minicomputer like the Perq.

(7) PDBIS users avoid a lot of typing and its inherent problems.

(8) The output subsystem, as it is explained in appendix D, adds another useful facility to the interface, and extends the use of the accessed data for further processing.

(9) The implementation of PDBIS has brought to light many approaches for its extension. PDBIS in its current status represents only a part of an integrated information system. The development of such a system will be achieved by implementing the ambitious plan for the future work, which is partially presented in this thesis and represented by:

(a) The implementation of the Output-File-Manager (OFM).

(b) The implementation of PDBIS in a distributed database environment.

(c) The implementation of PDBIS to access the CODASYL data model.

(d) The integration of PDBIS with other office information activities.

This plan for the future work provides an excellent research program in this area of computer applications.

(10) PDBIS is a portable system. It can be run under any operating system capable of operating a graphics package similar to GRAFIKS and supporting the use of

selection device (ICL-82b). In fact PDBIS is independent of the Perq operating system and the database management system that operates underneath. Therefore PDBIS can be run in most of the recent developed system such as the Lisa, Macintosh, IBM-PC, Atari, Amiga, *etc.* Writing the interface mainly in Fortran makes it very easy for the ordinary user to maintain and develop the interface, and to transfer it to other machines.

(11) It offers flexibility in query formulation, no matter what order is chosen to select relation or attribute names in any particular level of the query formulation. For instance, in Figure 4.1, if PART has to be joined with SUPPLIER in order to project PNAME and SNAME, it does not matter whether PART or SUPPLIER relation is selected first. Also the selection order of PNAME and SNAME makes no difference. This is one of the characteristics of an easy to use interface.

PART

P#	PNAME	COLOUR	WEIGHT	CITY

SUPPLIER

S#	SNAME	STATUS	CITY

Figure 4.1

In fact, the above enumerated properties become as important as the primary objectives of the PDBIS, because they represent the practical characteristics of the interface.

4.4. Perq

Perq (ICL-82b) is a powerful personal computer with a large internal data storage capacity and a very fast microcoded processor. It is equipped with a 24 MByte winchester disc drive as its mass storage device, and a 1 MByte floppy disc drive for backup. It presents information on a high resolution display of A4 size (long side vertical) in its standard font. The display can present a page of approximately 75

lines, each of up to 85 characters. Part or all the page can contain extremely detailed diagrams.

The Perq is supplied with a typewriter-style keyboard and a graphics tablet with a four button pointing device (the Puck), which can be used to input or construct diagrams, or point to any part of the display. By moving the pointing device (Puck) over the tablet, one can instantly indicate any part of the display screen. The picture on the display is known as a 'Raster' and consist of 1024 horizontal lines. Each line contain 768 picture elements (Pixels), each of which can be black or white. The selected part of the screen is indicated by a software cursor.

Each Perq machine is also supplied with a standard RS232C input/output interface for communication with serial devices such as printers, plotters, or other computers. GPIB (IEEE-488 standard) is also available in the Perq for communication with parallel devices like logic devices and some printers. In the PDBIS implementation, the Perq is connected with other machines through the serial line RS232C using the necessary software for transmitting and receiving information. These specification of the Perq are now available in a number of machines such as the IBM-PC, the Lisa, the Macintosh, the Atari, and many others which can implement PDBIS easily as well.

4.5. Graphics facilities

The graphics package which runs on the ICL-Perq computer system is called GRAFIKS. It is a toolkit of facilities for building the graphics systems, with particular provision for interactive programs. GRAFIKS is based on the principle of the ISO Standard Graphical Kernel System (GKS), and it is available on the Perq as a set of Fortran subroutines or Pascal procedures (ICL-82a).

PDBIS employs most of the facilities provided by this package to draw the interface layout on the screen, writing string characters and output text; it also

provides the interaction with the Puck, cursor control, and keyboard input. It has been used to open and close viewports and windows on the screen. The Puck position on the tablet returns data to the user in terms of the screen coordinate, which is rounded to the nearest pixel. Thus the tablet itself can be considered as another type of raster device. This technique is employed to map the selected element on the screen to its synonym name in PDBIS representation. In addition to the routines and procedures, the GRAFIKS system deals with two types of information files, one containing a set of fonts, and the other containing the GRAFIKS error messages.

4.6. Screen design

The screen is the most important component of any two dimensional language interface. It represents the communication bridge between the user and the system, and it is the means by which the idea of the interface is presented and through which it is implemented. Thus the design of the screen becomes one of the most important tasks the interface designer has to consider. A well-designed screen format can increase user processing speed, reduce user errors, and speed computer processing time. A poorly-designed screen will have the opposite effect, that is it will decrease user processing speed, provoke user mistakes, and complicate machine operations. A well-designed screen will increase user productivity, and a poorly designed one will degrade it.

The design of PDBIS screen is based upon a number of considerations. These considerations can be categorized as: Users, hardware, software, and applications (GALI-82).

User consideration are the needs and requirements of users and are oriented toward clarity, meaningfulness and ease-of-use.

Hardware and software considerations reflect the physical constraints of the Perq terminal; the screen size limits, and the characteristics of the controlling program. They provide a frame work within which the screen design must occur.

Application considerations reflect the objectives of the system for which the screen is being designed; they are the information building blocks which make up the screen.

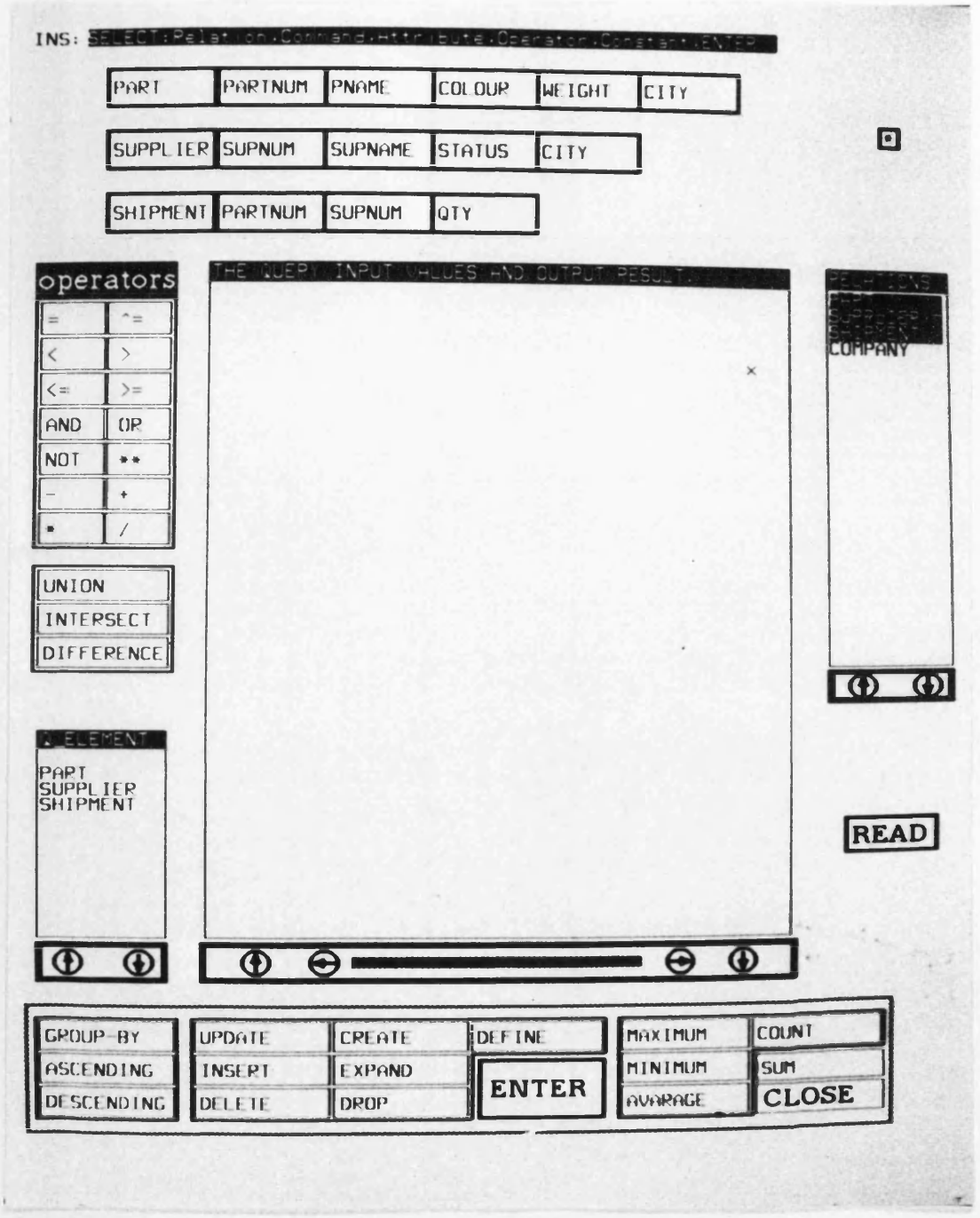

INS:
PART
PARTNUM
PNAME
COLOUR
WEIGHT
CITY
SUPPLIER
SUPNUM
SUPNAME
STATUS
CITY
SHIPMENT
PARTNUM
SUPNUM
QTY
operators
=
^=
<
>
<=
>=
AND
OR
NOT
**
-
+
*
/
UNION
INTERSECT
DIFFERENCE
THE QUERY INPUT VALUES AND OUTPUT RESULT
COMPANY
ELEMENT
PART
SUPPLIER
SHIPMENT
READ
GROUP-BY
ASCENDING
DESCENDING
UPDATE
INSERT
DELETE
CREATE
EXPAND
DROP
DEFINE
ENTER
MAXIMUM
MINIMUM
AVARAGE
COUNT
SUM
CLOSE

Figure 4.2

Figure 4.2 shows the PDBIS screen layout which is divided into five distinct areas. The main bulk of it represents the main display window which is located in the middle, and it has dimensions of (450 x 550) pixels. This window is defined by a GRAFIKS utility procedure call. Through this window the end user communicates with the system, the output results, error messages, and the formal query representation are displayed. The constant values are also entered into the system through this window. Any dialogue between the user and the system is carried out through this window. Size limitations of this window are avoided by using the scrolling technique. The output result can be scrolled up and down, left and right across the window. Thus, any part of the output can be viewed. Along the bottom of this window, there is a narrow rectangle containing four arrows in circles which indicate the scrolling direction. Scrolling is activated by pointing the cursor into any one of these circles and depressing the yellow button of the Puck.

The second area is the lower part of the screen, it is specified to contain an extensive commands and functions menu. Commands and functions are functionally grouped in this menu in order to be easily located. Elements of this menu represent the updating commands, ordering and group-by commands, ENTER and CLOSE commands and a number of built-in functions.

The third area represents the right hand side of the screen, where the relation names menu is located. This menu is not a static menu like the commands and functions menu, but is dynamic; the relation names in it can be scrolled up and down, by pointing to any one of the two arrows in the control box at the bottom of it, and these relation names can be updated according to the status of the database. Below this menu there is a small box containing the READ command.

The fourth area is the upper part of the screen. It is specified to be occupied by three relation skeletons where each skeleton can contain up to eight attributes at any

one time. It is possible to allow the name of each attribute in any skeleton to be scrolled left and right in order to display relations with higher degrees. The first cell in the skeleton represents the relation name. On the very top of the screen there is a one line instruction statement. A small box on the top right contains the asterisk character (*), the selection of which indicates that the whole relation attributes have to be projected.

The fifth area is the left hand side of the screen which contains three different menus; the top one contains all the arithmetic and logical operators, the one below contains the relational operators; UNION, INTERSECT, and DIFFERENCE. The bottom menu is a small window, containing the query elements which have been selected during the building operation of the query. These elements in the window can also be scrolled up and down to be viewed by the user. The purpose of this menu is to provide another way of query feedback to the user, which allows him to trace the query as a sequence of primitives displayed according to their selection order.

In recently developed systems, one can notice the common use of the multiwindowing technique where you can enlarge, create, move and overlap windows. This technique does not add any important advantage to PDBIS screen. The adoption of such technique in PDBIS requires further hardware and software resources, which certainly exhaust and slow the system. The provided scrolling facilities have satisfied the requirements without adding complexity to the system.

CHAPTER 5

PDBIS SPECIFICATION

5.1. PDBIS components

PDBIS components are the hardware devices, software utilities, and the user interaction. These components operate together to form an integrated interface capable of achieving specified objectives and showing certain properties. Figure 5.1 shows the main components of PDBIS.

(1) **User**: the user is any person who utilizes the interface for creating, retrieving, and updating the database. It does not matter whether he has used the computer directly or indirectly, or even if he has done any computer programming. The user is the element which activates the interface system. Unlike other computer programs which can be run in the absence of the user interaction, in PDBIS the user is actually a part of the system; his interaction with the system must continue until he gets the answer of his request.

(2) **Graphics utility**: this utility is represented by a set of interactive graphics routines and devices attached to the Perq machine. It provides a sketching facility and interaction means between the user and the system.

(3) **Perq operating system**: It represents the utility routines, operations controlling the attached devices, compilers and microcodes. These work jointly to manage and control the Perq machine and run the interaction facility between the real world (user, user program) and the database management systems through PDBIS.

(4) **Perq screen**: it is the two dimensional visual part of the Perq, which represents a work area where the real world is represented in the form of relations and attributes. In this area actions on the simulated real-world take place through

the query formulation process. The user monitors the success of his interactions with the DBMS in this area through the output result and error messages.

(5) **Controller**: the controller represents the intermediate piece of software between the Perq screen and the analyzer which controls the passage of the query after its formulation on the screen. The controller decides to which analyzer in the system the query should passed.

(6) **Analyzer**: a number of analyzer can be operating under PDBIS-screen manager, each of them represents a specific database management target. The analyzer is the main and most complicated part of the system. It receives the query primitives (PDBIS representation) from the controller and starts analyzing, interpreting, and translating this representation into a formal representation of the specified database language. This representation of the query is then transmitted through the serial line to another machine, where the DBMS executes the query and accesses the data.

(7) **Output-File-Manager (OFM)**: it is a software subsystem which receives the accessed data from the host computer and allows the user to specify the form of the output he wants before displaying it on the screen. The OFM is the topic of Appendix D in this thesis.

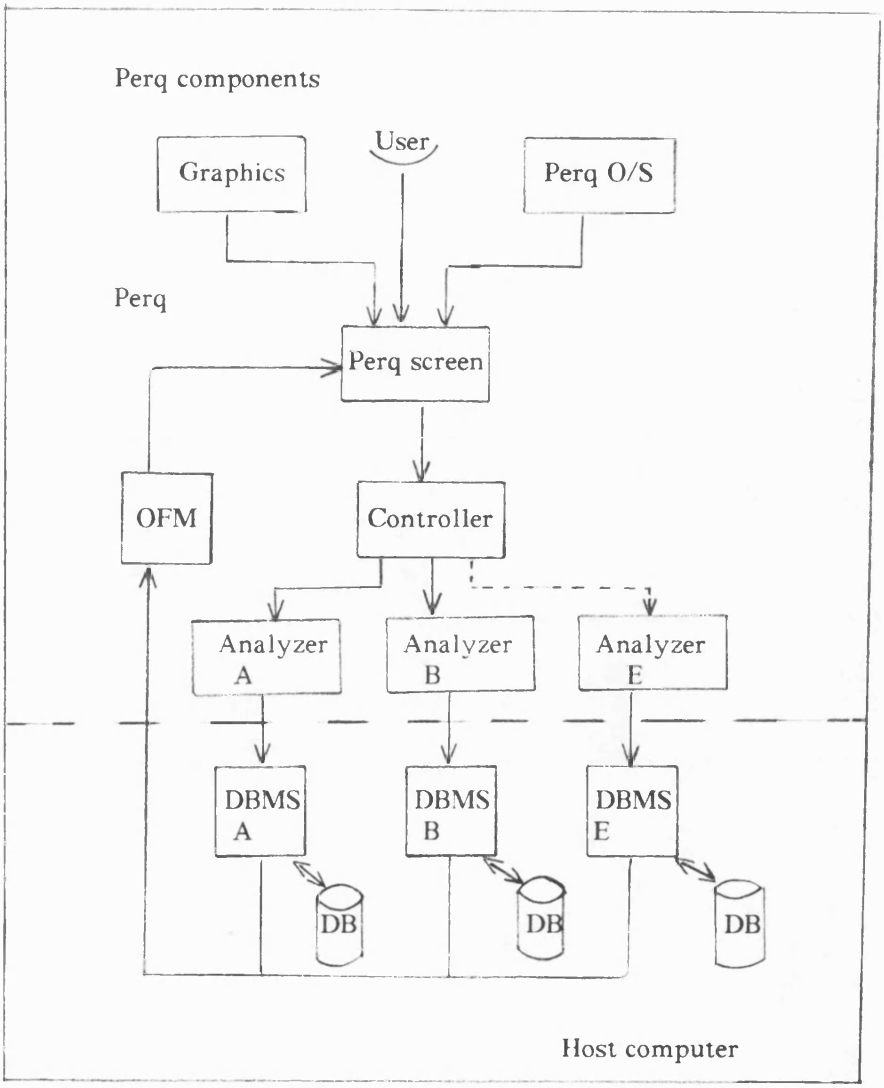


Figure 5.1 PDBIS configuration.

5.2. System layout and operations flow

Figure 5.1 also shows the system configuration and the operations flow among its components. The user constructs his query on the screen using graphics utilities which are controlled by the Perq operating system. Then this representation of the query is passed to the controller, where certain analysis is done, so that the controller can decide to which analyzer the query should be passed. The entire analysis and interpretation is carried out in the specified analyzer, in order to reconstruct the query in the formal representation of the specified underlying database query language. At

this stage a copy of this representation of the query is sent back to the user and displayed on the screen, while the original copy is transmitted to the host computer, where the specified DBMS processes the query and accesses the database. The result is sent back to the Perq screen through the Output-File-Manager which provides various output formats and report generation facilities. Error messages initiated by the DBMS are directly displayed to the user on the screen.

5.3. Query formulation

Throughout the rest of this thesis, the following three relation database is considered for the given examples. The relation PART (P) of degree 5 consists for each part, a part number, name, colour, weight, and location where the part is stored. The relation SUPPLIER (S) of degree 4 consists for each supplier, supplier number, name, status code, and location. The relation SHIPMENT (SP) of degree 3 consists for each shipment, a part number, supplier number, and the quantity shipped.

PART (PARTNUM PNAME COLOUR WEIGHT CITY)

SUPPLIER (SUPNUM SUPNAME STATUS CITY)

SHIPMENT (PARTNUM SUPNUM QTY)

The underlined elements indicate the primary keys in the relation.

The fundamental operation in SQL, LILA, and most other relational database sublanguages is the mapping operation, represented syntactically as a SELECT-FROM-WHERE block. This is used to SELECT attributes FROM one or more relations WHERE the tuples of the relations satisfy certain conditions. PDBIS consider another form of the mapping block, implicitly represented as FROM-SELECT-WHERE. This is also used to SELECT attributes FROM one or more relations, following the same order of the physical operations as expressed in the mapping block SELECT-FROM-WHERE, but the order of the logical operations within the block starts by specifying the

relations (FROM clause) instead of selecting the required attributes (SELECT clause). However, the mapping operation in both cases is a horizontal subsetting, (find all tuples which satisfy certain condition) followed by a vertical subsetting (extract the specified attributes from these tuples). In other words, it is a horizontal subsetting (usually an algebraic select), followed by a project operation. The FROM and SELECT clauses must always be specified in any mapping block. The WHERE clause may be omitted, in which case the SELECTed attributes of every tuple in the relation are extracted.

In PDBIS, the query formulation process follows the logical structure of operations in the mapping block. This logical structure is consistent with the user logic for the query construction, since he is thinking in terms of relations and attributes. For instance the query 'get all the supplier names' may be thought of as being constructed as follows: in order to project the supplier names, the user has to recognize first which relation contains them. PDBIS provides an easy way to help the user in this, while at the same time representing directly part of the query formulation process; all relation names in the database are available to the user on the screen, the structure of any relation can be simply displayed on the screen by selecting its name from the relation names menu, so that the user does not have to remember the relation names or their structure. This is really what the casual user needs. This style of presenting the database relations and their structure provides the user who is not familiar with the database structure a trial method to discover the relations structure and their relationships in the database. Thus, by selecting the right relation (supplier) (this is equivalent to a FROM clause), the user can then think of selecting the desired attributes (supplier name), which is equivalent to the SELECT clause in the mapping block. By this stage, any limitation on the projected result (which does not occur in this example) must be expressed by specifying the WHERE clause. The WHERE clause is constructed by the selection of the comparison operator followed by the selection of the two operands.

The above discussion of the formulation process of the retrieval query may be summarized by the following steps:

(1) Select relation names from the relation names menu (construct FROM clause).

(2) Select the attribute names from the relation skeletons (construct SELECT clause).

(3) Construct the WHERE clause, if there is any limitation in the selection of tuples.

Each one of the above points can be repeated any number of times depending on the query structure.

The following comparison operators may be used to express conditions in the WHERE clause:

>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
=	equal
^=	not equal

Any of these operators may be used with real or integer numbers. Comparisons may be made between an attribute and a constant or one attribute and another attribute. These operators may also be used with character constants. More than one condition (multiple condition) may be linked by the logical operators, OR and AND. The logical operator NOT may also be used.

In addition to comparisons between numbers, PDBIS allows the comparison of sets, as it is supported by the underlying database management systems. A set may be thought of as a collection of items, where each item is either a single attribute from a tuple, or a conglomerate of attributes from a tuple. Since a single attribute or collection of attributes can be regarded as tuples, the items which make up a set will be thought of as relations. Sets may be compared by asking whether set A contains every item in set B, in which case we say A contains B, or, does every member of A appear in

B, in which case we say A is in B. Sets may also compared for equality. To be able to compare two sets, the items of which the set consists should be of the same type, that is, they should have the same number of attributes, and should be possible to compare these attributes for equality. All set comparisons remove duplicate members from each set before the comparison operation is carried out.

The available set operators in PDBIS are:

UNION
INTERSECT
DIFFERENCE

These operators take two sets and return another set. The UNION of two sets consists of all members that belong to one or both of the sets. Example, (a, b, c) UNION (c, d, e) = (a, b, c, d, e). The INTERSECTION of two sets consist of those members belonging to both sets. Example, (a, b, c) INTERSECT (c, d, e) = (c). Set A DIFFERENCE set B consists of those members which belong to set A but not to set B. Example, (a, b, c) DIFFERENCE (c, d, e) = (a, b).

The arithmetic operations that are available in PDBIS are:

\+ addition
\- subtraction
/ division
* multiplication
** exponentiation

These operators may be used to construct arithmetic expressions for the selected attributes in the SELECT clause of the query.

PDBIS provides a number of built-in functions which can be used to provide a simple statistical and summary information, which enhances the retrieval power of the language.

These functions are:

AVG	---	average of argument values.
COUNT	---	counting of argument values.
SUM	---	totalling of argument values.
MAX	---	maximum of argument values.
MIN	---	minimum of argument values.

A relation may be partitioned into groups according to the value of some attributes, the SELECT clause is then applied to each group of the partitioned relation, rather than to each row of the original relation. Each expression in the SELECT clause must be single valued for each group. A condition may be applied to choose only certain of the groups and disqualify others. This type of condition will always use a built-in function, and is placed in a HAVING clause following the GROUP-BY clause.

In PDBIS both the GROUP-BY and GROUP-BY with HAVING clause are specified by the selection of the GROUP-BY command. The order in which data is displayed is controlled by using the ORDER-BY clause according to the values of the attributes specified in the ORDER-BY clause. In PDBIS the ORDER-BY clause is specified by the selection of the sorting order command, ASCE.or DESC.

PDBIS provides most of the update commands which are implemented in the SQL database language. Using PDBIS to express some of the updating commands requires a form of a dialogue between the user and the system. This dialogue act to clarifies the user intentions.

PDBIS provide the following updating commands:

UPDATE	: To modify a tuple in a relation.
INSERT	: To add new tuple to a relation.
DELETE	: To delete tuples or selected tuples from a relation.
CREATE	: To create a new relation.
EXPAND	: To expand an existing relation to include more domains.
DROP	: To drop a relation from the database.
DEFINE	: To define an external relation (view) from existing relations.

The READ command should be selected before typing any constant value.

ENTER is the last command to be selected for every transaction. It allows the query to be passed for further processing.

CLOSE is the command used to close the interface and return to the Perq system.

Details and examples of most of the above operations and functions are given later in this chapter.

5.4. Program specification

There are four main programs in PDBIS which perform the interface management, query formulation, query translation, and the output control. These programs are: the screen manager, the controller, the analyzer, and the Output-File-Manager.

5.4.1. Screen-Manager

This is the main program in PDBIS which is executed once the user types PDBIS after a successful login operation in the Perq. A security level for using the interface is provided here; the system asks the user to provide a special password, database name, and the database management system that he intends to use. Then the program issues the necessary commands to open the database in the host computer and retrieve all the relation names in the specified database to be displayed in the relations menu on the Perq screen. Then the Screen-Manager program performs the following functions sequentially:

(1) Sketches the screen and fills in the menus with the specified text.

(2) Sets up the selection devices by activating the Puck buttons and its location on the tablet.

(3) Manages the process of creating and destroying the viewports that are specified to contain the selected relation skeletons.

(4) Retrieves the attributes of the selected relations from the database and fits them in the specified skeletons.

(5) Provides the scrolling facility in the relation menu, main window, and in the query primitives window.

(6) Controls the prompts of the instruction statement which is displayed on top of the screen. This statement consists of a number of options that the user may choose to select. The instruction statement takes the following forms through the query formulating process:

Select Relation, Command.
Select Relation, Attribute, Command.
Select Relation, Attribute, Command, Operator, ENTER.
Select Relation, Attribute, Command, Operator, constant value, ENTER.

The last form of the instruction statement covers any possible selection that might be required in any level of the query formulation process. This statement will remain on the screen after it has been displayed until the ENTER command is selected.

The screen manager program maps the selected elements on the screen into the intermediate representation of the query (PDBIS representation). The selected query primitives on the screen return to the Screen-Manager program as X, Y coordinates. The program scans these coordinates in the PDBIS primitives directory and finds their synonym elements to be stacked in a two dimensional array.

5.4.2. Controller

The controller is a subroutine which is activated by the Screen-Manager program after the query is formulated. Its main functions are to select the right analyzer and specify the parameters that are to be passed to that analyzer. The controller receives

the intermediate representation of the query in the form of a two dimensional array. For example the query 'get part numbers for parts supplied by S2' is received by the controller in the following form:

SHIPMENT	T1	--
PARTNUM	T2	SHIPMENT
=	T3	--
SUPNUM	T2	SHIPMENT
S2	T10	--
ENTER	--	--

In the above representation the first column contains the names of the selected elements (the query primitives). The second column contains the group type of the selected elements (the elements on the PDBIS screen are classified into a number of groups according to their functions and locations on the screen). The third column contains the relation names for the selected attributes.

The controller also receives through its arguments the attributes of the seleced relations stored in one dimensional arrays. The form of data structure of the query representation will not be changed in the controller. After some analysis the required parameters for the analyzer are specified and the chosen analyzer is activated.

5.4.3. Analyzer

The analyzer is that part of the system which performs the analysis of the PDBIS representation of the query, translates this representation into formal statements of the underlying database language, and submits the query to the specified DBMS. Though all analyzers employ the same basic structure and purposes a different analyzer is required for each DBMS. The complexity of each analyzer design and implementation is directly related to the complexity of the query language of the DBMS, and also to the facilities provided by that language. A comparison of the complexity of the four analyzers, which have already been developed in the current implementation of PDBIS and presented in detail in this chapter and in the appendices

A, B, and C, is shown by the difference in the DBMS formal query statements for the query, 'Find all the supplier numbers who supply part P2.'

The SQL query would be:

```
SELECT SUPNUM
  FROM SHIPMENT
    WHERE PARTNUM = 'P2'
```

The LILA query would be:

```
SELECT SUPNUM
  FROM SHIPMENT
    WHERE PARTNUM = "P2"
```

The SALT query would be:

```
LIST SHIPMENT: PARTNUM = 'P2' % SUPNUM
```

While the query in DBASE-II would be:

```
USE SHIPMENT
LIST SUPNUM FOR PARTNUM = 'P2'
```

All these languages are considered as a non-procedural query languages, but SQL and LILA are more structural than SALT and DBASE-II. The differences in the complexity may be not very obvious in a simple query like the above given one, but in more complicated ones the difference will become more clear. However, the four analyzers consider the same basic techniques in analyzing the query and in the construction of the formal DBMS language statements. As it has been mentioned before, the query formulation process depends conceptually on the mapping operation, represented by the mapping block 'FROM-SELECT-WHERE'. The analyzer begins its operation by passing the PDBIS representation of the query through a series of lexical analysis operations, in order to identify distinctly each clause in the query as well as any relational operation which might be involved. The analysis is done by making top to bottom passes through the three columns of the two dimensional array which holds the query. A number of rules are applied during the passes by which operations such as selection, projection, and join are identified and the three mapping clauses are

constructed separately. These rules might be applied a number of times for one query, depending on the complexity of the query structure. One can expect that a number of join operations, a number of selection operations, and a number of conditions might be involved in the same query. In addition, a WHERE clause may consist of one or more subqueries which might contain nested clauses as well.

In the analysis process, the intermediate representation of the query is treated as a number of binary trees. Each tree represents one clause of the main mapping block or a function clause such as GROUP-BY or ORDER-BY. Through the preorder traversing process on each tree predefined rules are applied to construct the clause which is represented by that tree. Examples of these rules are:

(1) A number of successive relation nodes followed by a number of attribute nodes, map to a projection of a number of attributes out of a join operation between a number of relations. This later maps to a SELECT and FROM clause in the translation process.

(2) One relation node followed by a number of successive attribute nodes, maps to a projection operation for a number of attributes out of a selected relation. This also maps to a SELECT and FROM clause in the translation process.

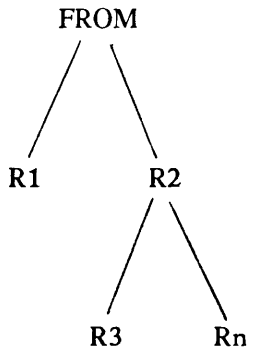
(3) The operator node indicate the start of a condition clause. The following nodes represent the operands of the condition. These operands might be one or more element targets represented directly by constant values or produced by one or more subqueries.

(4) GROUP-BY, ASCE, DESC, built-in functions, *etc.* nodes indicate the existence of that particular function in the query. The following nodes represent the arguments of that function. These arguments might be a constant value, or a constant value with conditional clauses or a constant value produced by nested subqueries.

(5) The updating command represented by the first node of the tree identifies an updating query. The analysis of the updating query follow the same analysis technique of the retrieving query.

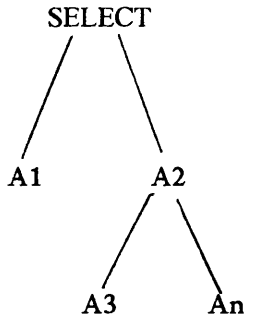
The following represents the general structures of the main clauses in the mapping block as seen by the analyzer. The details of these structures and other functional clauses structures are explained in section 5.5 through the presentation of SQL analyzer and its implementation. The details of other implemented analyzers are explained with examples in the appendices A, B, and C.

FROM clause

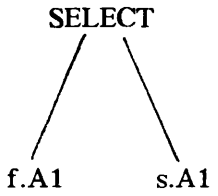


The selection can be done from one or more relations. If the selection is made from more than one relation a join operation must occur among them.

SELECT clause



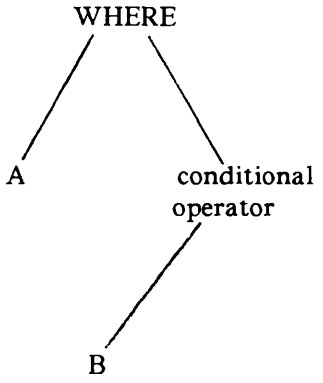
One or more attributes can be projected out of one or more relations in one single query. If the selection is made from two identical relations, this means that two different tuples are selected from a relation joined with itself.



(f) and (s) are two arbitrary names used to distinguish between the appearances, and they are used as qualifiers in SELECT and FROM clauses.

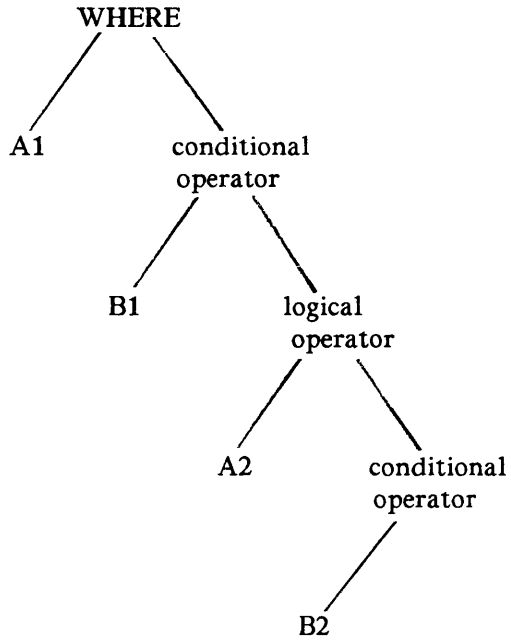
The WHERE clause is more complicated than the SELECT and FROM clauses, because it may take a variety of structures, and includes many combinations. Each of the implemented analyzers have a different set of WHERE clause structures, but the general structure of the clause can be introduced as:

WHERE (A) conditional operator (B).



'A' and 'B' may represent either an attribute type or a constant value. A constant value can be a real, integer, or character, as well as a single value or a set of values.

A combination of conditions (multiple condition) may be found in the same query. These conditions are combined by the logical operators, AND, OR, and NOT. This type of condition structure is shown as following:



5.4.4. Output-File-Manager

The accessed data from the database is not only used for viewing, but it is also used as source data for further processing to achieve certain purposes. However, whatever the reason for accessing the data, it has to be presented to the user, or for further processing, in a useful format.

Most of the present end-user interfaces present the accessed data to the user in one tabular format, which is the standard DBMS format. Therefore it is very important to have a control utility for the output format as a part of the data manipulation language or the end-user interface. Accordingly, PDBIS is provided with the Output-File-Manager (OFM). It is an easy to use facility, easy to be implemented, and it is compatible with the use of PDBIS. This utility program allows the user to have the control of his data, and specify the format which is suitable for the purpose for which the data was accessed. It maximizes the efficiency of the database by better utilizing the accessed data. This utility program can also be a channel for PDBIS to be used with other data processing systems. It makes PDBIS a multifunctional interface,

which allows the same user to query the database, produce the desired output format, and generate external reports. This will increase the productivity and affect the cost, as fewer people will be required to do these functions. Presenting the manager with the data he wants, in the format that facilitate the understanding and the analysis of the problem will certainly support his decision making processes. The OFM also provides commands to save the query and the output result for later use. The details of the OFM usage is given in Appendix D of this thesis.

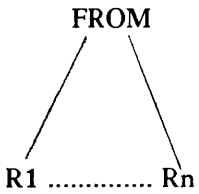
5.5. SQL analyzer

The SQL analyzer consist of two sets of subroutines; the first set is concerned with the analysis and the translation of the retrieving queries to SQL representation, while the second set is integrated functionally with the first set of subroutines to perform the analysis and the translation of the updating queries.

The target of the analysis process for any retrieving query and most of the updating ones is achieved by the construction of the three basic clauses of the mapping block 'FROM-SELECT-WHERE'. Therefore, the specification of the analyzer is determined by the syntax structure of these clauses in it. In this section the syntax formats of each clause supported by SQL-analyzer are presented in the form of binary trees.

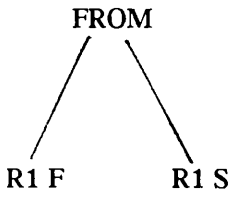
(1) **FROM clause**:

Format (1):



In this format the maximum number of relations which can be selected at any one level of the query is five, (n=1, 2, . . , 5). This number is restricted by the number of viewports which can be created in the available memory.

Format (2):



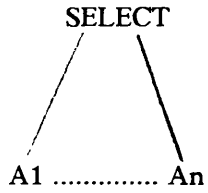
This format represents the selection from a relation joined with itself.

(2) SELECT clause:

Format (1):

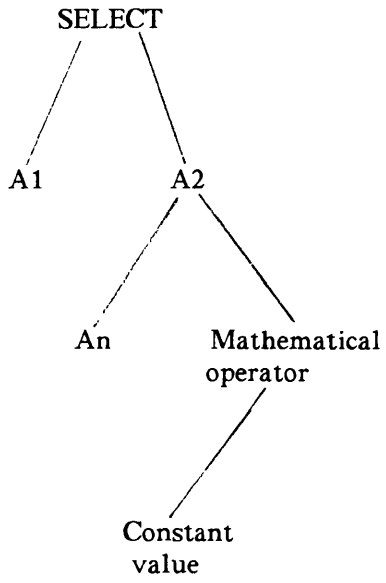
The asterisk means that all the attributes of the selected relations in FROM clause are to be projected.

Format (2):



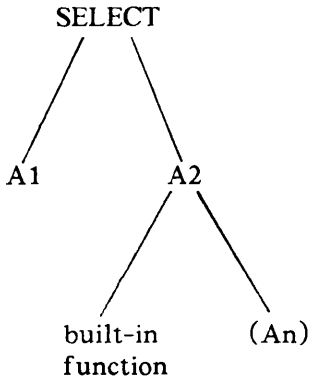
In the current implementation of this format, up to eight attributes can be projected from the selected relations, at any one level of the query. This has to be extended to an unlimited number of attributes, after providing the left, right scrolling facility to the relation skeleton.

Format (3):



In this format the last selected attribute may be applied to any mathematical operation.

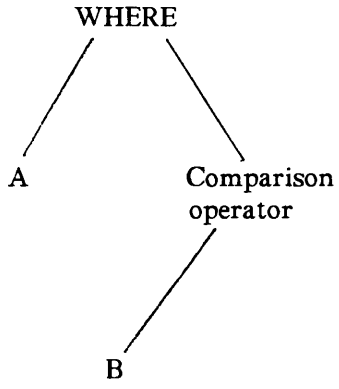
Format (4):



This format shows that, a built-in function can be applied to the last selected attribute.

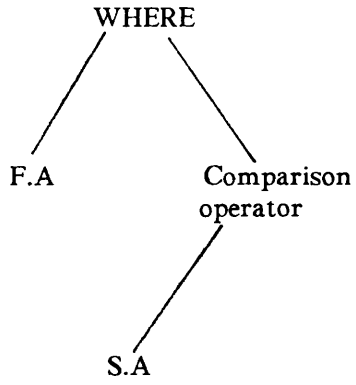
(3) **WHERE clause**:

Format (1):



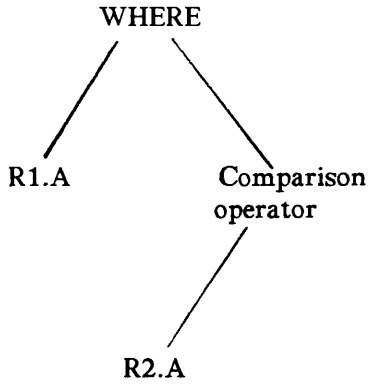
'A' represents any attribute name in the selected relations, and 'B' represents any constant value.

Format (2):



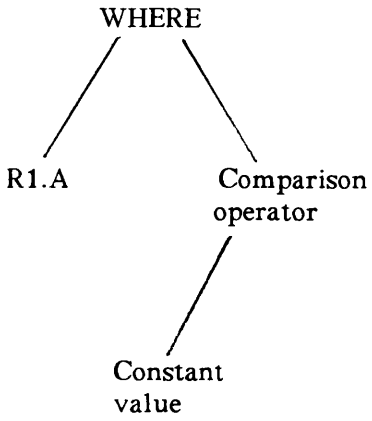
'F.A' and 'S.A' are two values for the same attribute in two different tuples in one relation.

Format (3):



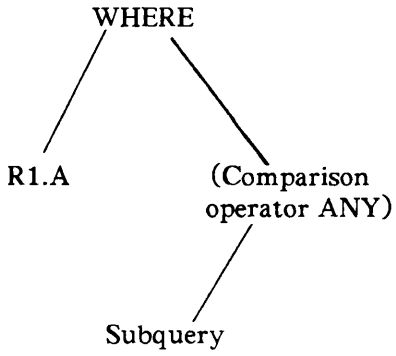
'R1.A' and 'R2.A' are the same attribute name in two different relations.

Format (4):



'A' is a common attribute name in relation R1 and other relations joined with R1. 'R1.A' indicates the value of the attribute A in the relation R1 to be compared with the constant value.

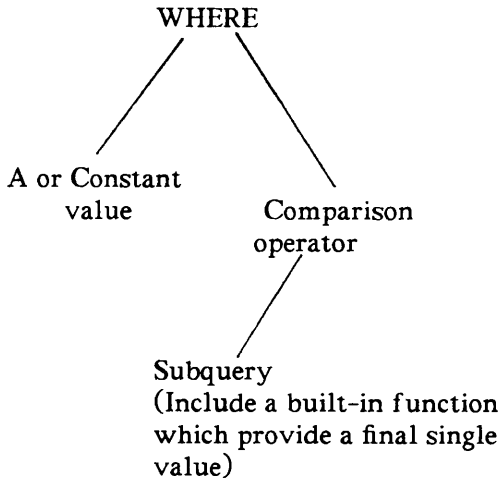
Format (5):



In this structure the clause represents a condition in which a value of attribute A in relation R1 is compared with a set of values generated by a subquery.

The comparison operator (Comparison operator plus ANY) is interpreted as: the condition F (=, ≠, >, <, ≤, ≥) ANY ({s1, s2, s3. . . }) evaluates to true if and only if the value F is (=, ≠, >, <, ≤, ≥) to at least one value in the set {s1, s2, s3, . . .}.

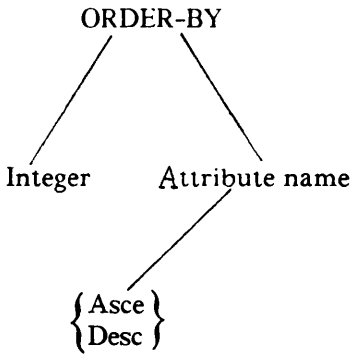
Format (6):



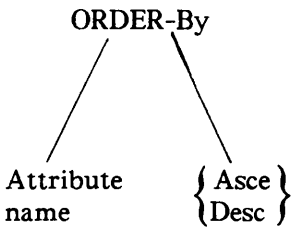
This format represents a condition in which an attribute value or a constant value is compared with one single value produced by a subquery which includes a built-in function.

(4) **ORDER-BY clause**:

Format (1):



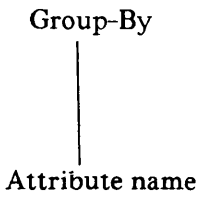
Format (2):



In the above two formats, {Asce, Desc} means that one of the options has to be selected to specify the sorting order of the projected attributes.

(5) **GROUP-BY clause**:

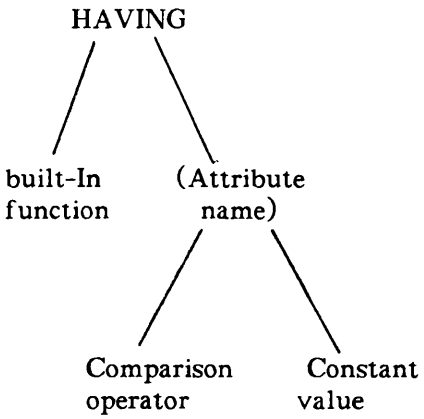
Format (1):



This format represents the GROUP-BY clause which consist of the GROUP-BY command and an attribute name as an argument.

(6) **HAVING clause**:

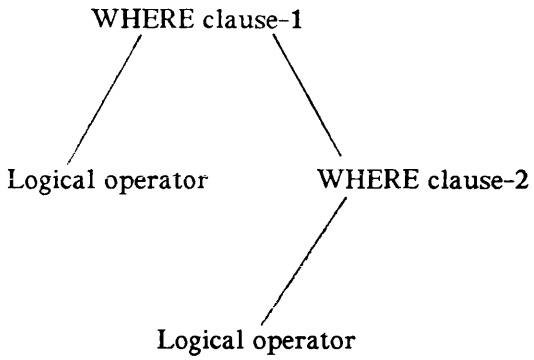
Format (1):



In fact the HAVING clause represents a WHERE clause, its left operand represents a single value produced by applying a built-in function to one of the selected attributes. The right operand represents a single value, which may be specified directly as a constant value or produced by a nested subquery.

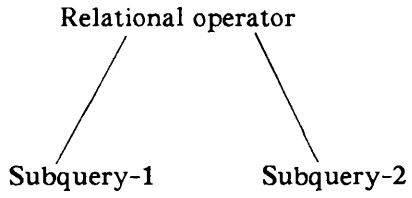
Having clause is usually follows the GROUP-BY clause.

(7) **Logical operator format**:



This format represents a number of WHERE clauses linked together with logical operators, and can be repeated for a number of times in one query.

(8) **Relational operator format**:



In this format the relational operator is applied to two sets of data values generated by subquery-1 and subquery-2. The result will be a third set of data values.

5.6. PDBIS-SQL Interface

The properties of PDBIS-SQL interface is presented in this section through a variety of retrieving and updating examples supported by the necessary explanation. These examples will also demonstrate the way in which PDBIS is used to construct various queries.

In the following PDBIS representation the arrows indicate the selected elements of the query on the Perq screen, and the dotted lines specify the typed constant values.

5.6.1. Retrieving operations

Q1: Get full details for all suppliers.

PDBIS representation	SQL representation
--------> SUPPLIER	SELECT *
--------> *	FROM SUPPLIER
--------> ENTER	

Q2: Get part numbers for all parts supplied.

--------> SHIPMENT	SELECT PARTNUM
--------> PARTNUM	FROM SHIPMENT
--------> ENTER	

PDBIS does not support duplicate elimination from the result as the SQL does via the keyword UNIQUE, because this operation is costly and users will not usually be bothered by the presence of duplicate in their output. However, other DBMSs like CODD implicitly eliminate duplication from the output result.

Q3 : Get supplier numbers for suppliers in Paris (photo SQL1).

--------> SUPPLIER	SELECT SUPNUM
--------> SUPNUM	FROM SUPPLIER
--------> =	WHERE CITY = 'Paris'
--------> CITY	
..... 'Paris'	
--------> ENTER	

In the above query the WHERE clause is introduced. It starts by selecting the comparison operator followed by the attribute name which has to be satisfied with the second operand 'Paris'.

Q4: Get supplier numbers for suppliers in Paris with status < 50.

```
--------> SUPPLIER          SELECT SUPNUM
--------> SUPNUM            FROM SUPPLIER
--------> =                 WHERE CITY = 'Paris'
--------> CITY              AND STATUS < 50
..... 'Paris'
--------> AND
--------> <
--------> STATUS
..... 50
--------> ENTER
```

The above query shows two conditions, linked by the logical operator 'AND'. Both of the conditions have to be satisfied in order to project the supplier numbers.

Q5: Get supplier numbers for suppliers in Paris, in descending order of status (photo SQL2).

```
--------> SUPPLIER          SELECT SUPNUM
--------> SUPNUM            FROM SUPPLIER
--------> =                 WHERE CITY = 'PARIS'
..... 'Paris'               ORDER BY STATUS DESC
--------> DESC
--------> STATUS
--------> ENTER
```

In the above query the ORDER-BY clause is introduced; it is constructed by the selection of the sorting order 'DESC' first, then the attribute name (STATUS) to give the values by which the supplier numbers will be ordered.

Q6: For each part supplied, get part numbers and names of all cities supplying the part.

```
--------> SHIPMENT          SELECT PARTNUM, CITY
--------> SUPPLIER          FROM SHIPMENT, SUPPLIER
--------> PARTNUM           WHERE SHIPMENT.SUPNUM = SUPPLIER.SUPNUM
--------> CITY
--------> =
--------> SUPNUM (in SHIPMENT)
--------> SUPNUM (in SUPPLIER)
--------> ENTER
```

In the above query PARTNUM and CITY, required to be projected out of a relation

formulated by a join operation between the relation SHIPMENT and the relation SUPPLIER, over the attribute SUPNUM.

When two relations share a common data-item type they may be joined. The join operation puts together columns from different relations to form a new relation of interest from those columns. Generally speaking when relations are joined on a certain data-item type, only those tuples which share the same value of that data-item appears in the result without duplication. Any join operation in which the joining condition is based on equality between values in the common column is called equijoin. This type of join is the most frequently used. The equijoin necessarily contains two identical columns. It is possible to eliminate one of these two columns by applying a PROJECT operation. An equijoin with one of the identical columns eliminated is called a natural join.

In PDBIS, whenever there are two or more successive selections of relations, a join operation will take place during the query processing, and obviously that join operation depends on the join techniques used in the underlying database management system.

The join mechanism in the above example can be viewed in its simple form as : (a) The cartesian product of SHIPMENT and SUPPLIER is formed. (b) Tuples not satisfying join condition SHIPMENT.SUPNUM = SUPPLIER.SUPNUM are eliminated from the result of step (a). (c) Columns PARTNUM and city are extracted from the result of step (b). (d) Redundant duplicate tuples are eliminated from the result of step (c).

As was mentioned earlier, the number of relations which can be selected during the query construction is only five. But only three of them can be visible at one time. Therefore, a scrolling facility should be provided to enable the user to select from any one of the selected relations at any stage of the construction. Consequently, PDBIS

allows a join operation between up to five relations with specified attributes projected out of them. Practically this limit is sufficient, because the query which involves a join operation between more than three relations is considered to be costly, and so it is worthwhile to split such a query into simpler ones, or even to consider restructuring the database in order to avoid the complicated join queries.

Q7: Get all pairs of supplier numbers such that the two suppliers are located in the same city (photo SQL3).

```
--------> SUPPLIER                          SELECT F.SUPNUM, FF.SUPNUM
--------> SUPPLIER                          FROM  SUPPLIER F, SUPPLIER FF
--------> SUPNUM (in the first SUPPLIER)    WHERE F.CITY=FF.CITY
--------> SUPNUM (in the second SUPPLIER)   AND F.SUPNUM ^= FF.SUPNUM
--------> =
--------> CITY (in the first SUPPLIER)
--------> CITY (in the second SUPPLIER)
--------> AND
--------> ^=
--------> SUPNUM (in the first SUPPLIER)
--------> SUPNUM (in the second SUPPLIER)
--------> ENTER
```

The above example shows how a relation is joined with itself. The join condition is that the two cities are the same (the join is done over the common attribute CITY). The second condition (the first supplier number is not equal to the second supplier number) will ensure that two different suppliers (pairs) will be listed. In this query the same relation was displayed twice on the screen, and SUPNUM was selected from each of them, as if they were selected from two different relations. For this case SQL introduced arbitrary names F and FF, and used them as qualifier in the SELECT and WHERE clause.

Q8: Get supplier names for suppliers who supply part P2 (photo SQL4).

```
--------> SHIPMENT              SELECT SUPNAME
--------> SUPPLIER              FROM SUPPLIER, SHIPMENT
--------> SUPNAME               WHERE SUPPLIER.SUPNUM = SHIPMENT.SUPNUM
--------> =                     AND SHIPMENT.PARTNUM = 'P2'
--------> SUPNUM (in SUPPLIER)
--------> SUPNUM (in SHIPMENT)
--------> AND
--------> =
--------> PARTNUM (in SHIPMENT)
..... 'P2'
--------> ENTER
```

The above query involves a join between two relations SHIPMENT and SUPPLIER, over the common attribute SUPNUM. The projection condition which has to be satisfied is that PARTNUM = P2. In this query, after the two relations are inspected, the result is entirely extracted from a single relation SUPPLIER. Therefore, this query may be expressed in another way (photo SQL5):

```
--------> SUPPLIER                SELECT SUPNAME
--------> SUPNAME                 FROM SUPPLIER
--------> =                       WHERE SUPNUM = ANY (
--------> SUPNUM (in SUPPLIER)                SELECT SUPNUM
--------> SHIPMENT                            FROM SHIPMENT
--------> SUPNUM (in SHIPMENT)               WHERE PARTNUM = 'P2')
--------> =
--------> PARTNUM (in SHIPMENT)
..... 'P2'
--------> ENTER
```

In this representation of the query, the expression in parentheses is a subquery which evaluates to a set of supplier numbers that correspond to part number P2. Then, SUPNAME is extracted from those tuples of the relation SUPPLIER, which their SUPNUMs exist in that set.

Q9: Get supplier numbers with status value greater than the current maximum value in the relation SUPPLIER (photo SQL6).

```
--------> SUPPLIER          SELECT SUPNUM
--------> SUPNUM            FROM SUPPLIER
--------> >                 WHERE STATUS > (SELECT MAX(STATUS)
--------> STATUS                            FROM SUPPLIER)
--------> SUPPLIER
--------> STATUS
--------> MAX
--------> ENTER
```

In the above query the WHERE clause include a subquery which provide one single value to satisfy the status. This single value of the status is obtained by applying the built-in function 'MAX' to the STATUS in relation SUPPLIER. In this case the WHERE clause produce an empty set, because there is no value greater than the maximum value.

In PDBIS the built-in function is employed by the selection of the function immediately after its argument.

The above query may be constructed by another way, without using the built-in function.

```
--------> SUPPLIER          SELECT SUPNUM
--------> SUPNUM            FROM SUPPLIER
--------> >                 WHERE STATUS > ANY (
--------> STATUS                          SELECT STATUS
--------> SUPPLIER                        FROM SUPPLIER)
--------> STATUS
--------> ENTER
```

In the above representation (> ANY) is used as the comparison operator which evaluates to true if and only if the status is greater than some value produced by the subquery in the clause.

Q10: Get supplier names for suppliers who supply at least one red part (photo SQL7).

```
--------> SUPPLIER                    SELECT SUPNAME
--------> SUPNAME                     FROM SUPPLIER
--------> =                           WHERE SUPNUM IN
--------> SUPNUM (in SUPPLIER)                (SELECT SUPNUM
--------> SHIPMENT                              FROM SHIPMENT
--------> SUPNUM (in SHIPMENT)                 WHERE PARTNUM IN
--------> =                                       (SELECT PARTNUM
--------> PARTNUM (in SHIPMENT)                   FROM PART
--------> PART                                    WHERE COLOUR = 'Red'))
--------> PARTNUM (in PART)
--------> =
--------> COLOUR
..... 'Red'
--------> ENTER
```

This query consist of a number of nested subqueries. It uses IN as an equivalent operator to (= ANY). Even though the above query in both SQL and PDBIS representation, looks as a long column of selected elements, its construction in PDBIS still can be achieved very quickly , providing that the user knows the logic of the question.

Q11: Get part numbers for parts that either weigh more than 18 pounds or are currently supplied by supplier S2 (or both) (photo SQL8).

```
--------> PART                SELECT PARTNUM
--------> PARTNUM             FROM PART
--------> >                   WHERE WEIGHT > 18
--------> WEIGHT
..... 18                      UNION
--------> UNION
--------> SHIPMENT            SELECT PARTNUM
--------> PARTNUM             FROM SHIPMENT
--------> =                   WHERE SUPNUM = 'S2'
--------> SUPNUM
..... 'S2'
--------> ENTER
```

This example introduce the use of set operator UNION. This operator separate two subqueries, each of them produce a set of values. The UNION between these two sets produces the result set which contains all the values that belong to one or both of them.

Q12: For all parts get part number and weight of that part in grams.

```
--------> PART                SELECT PARTNUM, WEIGHT*454
--------> PARTNUM             FROM PART
--------> WEIGHT
--------> *
..... 454
--------> ENTER
```

This example shows the use of the arithmetic operations which are usually applied to the last extracted attribute.

Q13: Get the total number of suppliers.

```
--------> SUPPLIER            SELECT COUNT(*)
--------> *                   FROM SUPPLIER
--------> COUNT
--------> ENTER
```

Q14: Get the total number of suppliers currently supplying parts.

```
--------> SHIPMENT            SELECT COUNT(SUPNUM)
--------> SUPNUM              FROM SHIPMENT
--------> COUNT
--------> ENTER
```

Q15: Get the total quantity of part p2 supplied.

```
--------> SHIPMENT            SELECT SUM(QTY)
--------> QTY                 FROM SHIPMENT
--------> SUM                 WHERE PARTNUM = 'P2'
--------> =
--------> PARTNUM
..... 'P2'
--------> ENTER
```

Q16: For each part supplied, get the part number and the total quantity supplied of that part (photo SQL9).

```
--------> SHIPMENT          SELECT PARTNUM, SUM(QTY)
--------> PARTNUM           FROM SHIPMENT
--------> QTY               GROUP BY PARTNUM
--------> SUM
--------> GROUP
--------> PARTNUM
--------> ENTER
```

The last four examples have shown the use of the built-in function applied to one of the projected elements. The last query shows the use of GROUP-BY function. In this query the relation SHIPMENT is divided into groups according to the value of PARTNUM, and then, the built-in function SUM is applied to each group.

Q17: Get part numbers for all parts supplied by more than one supplier.

```
--------> SHIPMENT          SELECT PARTNUM
--------> PARTNUM           FROM  SHIPMENT
--------> GROUP BY          GROUP BY PARTNUM
--------> PARTNUM           HAVING COUNT (*) > 1
--------> >
--------> *
--------> COUNT
..... 1
--------> ENTER
```

The above query shows the use of HAVING clause, which represent a condition specified after GROUP-BY clause. This condition always uses a built-in function and is applied to the groups that are formed by GROUP-BY clause.

Q18: For all parts such that the total quantity supplied is greater than 300 (excluding from the total shipment for which the quantity is less than or equal 200), get the part numbers and the maximum quantity of the part supplied; and order the result by descending part number within those maximum quantity values (photo SQL10).

--------> SHIPMENT
--------> PARTNUM
--------> QTY
--------> MAX
--------> >
--------> QTY
..... 200
--------> GROUP BY
--------> PARTNUM
--------> >
--------> QTY
--------> SUM
..... 300
--------> DESCENDING
..... 2
--------> PARTNUM
--------> ENTER

```
SELECT PARTNUM, MAX(QTY)
FROM SHIPMENT
WHERE QTY > 200
GROUP BY PARTNUM
HAVING SUM(QTY) > 300
ORDER BY 2, PARTNUM DESC
```

The above query combines the use of a number of functions that are available in SQL.

5.6.2. Update operations

PDBIS provides the main update operations which are supported by SQL-DML, or SQL-DDL. In PDBIS, the construction of updating queries may require the user to answer certain questions conducted by the system.

1. **UPDATE**

The query which involve UPDATE operation is constructed by the following steps:

(a) Select UPDATE command.
(b) Select the relation to be updated.
(c) Select the attribute to be updated in the selected relation.
(d) Enter any condition under which the elements to be updated are specified.

Then, the system will acquire the new value of the specified elements.

Q19: Change the colour of part P2 to yellow, increase its weight by 5, and set its city to 'unknown' (null) (photo SQL11).

```
--------> UPDATE                    UPDATE PART
--------> PART                      SET COLOUR = 'Yellow'
--------> COLOUR                    WEIGHT = Weight+10
--------> WEIGHT                    CITY= Null
--------> CITY                      WHERE PARTNUM = 'P2'
--------> =
--------> PARTNUM
..... 'P2'
--------> ENTER

ENTER NEW COLOUR:'Yellow'
-------------------------
ENTER NEW WEIGHT: Weight+10
-------------------------
ENTER NEW CITY: Null
---------------------
```

The underlined elements are displayed by the system to be answered by the user.

2. **INSERT**

The INSERT command is used to insert a single tuple in an existing relation, or it is used to insert a new tuple in which not all fields have known values.

Q20: Add part P7 (name 'Washer', colour 'Gray', weight 2) to relation PART (photo SQL12).

```
--------> INSERT
--------> PART
--------> PNAME      INSERT INTO PART: (PNAME,COLOUR,WEIGHT):
--------> COLOUR                       <'Washer','Gray',2>
--------> WEIGHT
--------> ENTER

ENTER PNAME VALUE: 'Washer'
---------------------------
ENTER COLOUR VALUE: 'Gray'
--------------------------
ENTER WEIGHT VALUE: 2
---------------------
```

3. **DELETE**

DELETE is the command which is used to delete a single tuple in a relation or all the tuples in a relation, as in the following examples.

Q21: Delete supplier S2.

```
--------> DELETE
--------> SUPPLIER          DELETE SUPPLIER
--------> =                 WHERE SUPNUM = 'S2'
--------> SUPNUM
..... 'S2'
--------> ENTER
```

In the above example a single tuple, S2 is deleted.

Q22: Delete all suppliers.

```
--------> DELETE
--------> SUPPLIER          DELETE SUPPLIER
--------> ENTER
```

Here the entire relation is deleted and becomes empty, but it is still known as a relation in the database.

4. **DROP**

All tuples in a specified base relation can be deleted by using DROP command. All indexes and views on that table are destroyed and its storage space is released.

Q23: Delete relation supplier.

```
--------> DROP            DROP TABLE SUPPLIER
--------> SUPPLIER
--------> ENTER
```

5. **CREATE**

CREATE is part of SQL-DDL command language, but it can be used at any time to create a new empty base relation with new attributes. This new relation may then

be filled by data values by using the INSERT command or by invoking the System-R loader (DATE-81), a system utility program, to perform this function.

Q24: Create relation COMPANY with attributes; CONUM, CONAME, and LOCATION.

```
--------> CREATE                CREATE TABLE COMPANY (CONUM (INTEGER)
--------> ENTER                                      CONAME (CHAR(10)),
                                                     LOCATION (CHAR (10)))

ENTER RELATION NAME: COMPANY
----------------------------
FIELD-1 IDENTIFICATION: CONUM (INTEGER)
----------------------------
FIELD-2 IDENTIFICATION: CONAME (CHAR (10))
----------------------------
FIELD-3 IDENTIFICATION: LOCATION (CHAR (10))
----------------------------
FIELD-4 IDENTIFICATION: END
----------------------------
```

The system will proceed to ask for the next field definition until the user enters 'END'. Then the query is processed, and if it is successful, a new relation will be created.

The field definition consists of field name and data-type.

The permissible data-types in SQL are:

CHAR (n): fixed length character string.

CHAR (n) VAR: variable-length character string.

INTEGER: fullword binary integer.

SMALLINT: halfword binary integer.

FLOAT: doubleword floating-point number.

6. **EXPAND**

An existing database relation can be extended at any time by adding a new column (field) at the right side of the relation. All the existing tuples of the expanded

relation are expanded with new field of value null.

Q25: Expand relation PART, by adding DATE.

```
--------> EXPAND                EXPAND TABLE PART
--------> PART                  ADD FIELD DATE (CHAR (6))
--------> ENTER

FIELD-1 NAME: DATE (CHAR(6))
----------------
FIELD-2 NAME: END
----------------
```

7. **DEFINE**

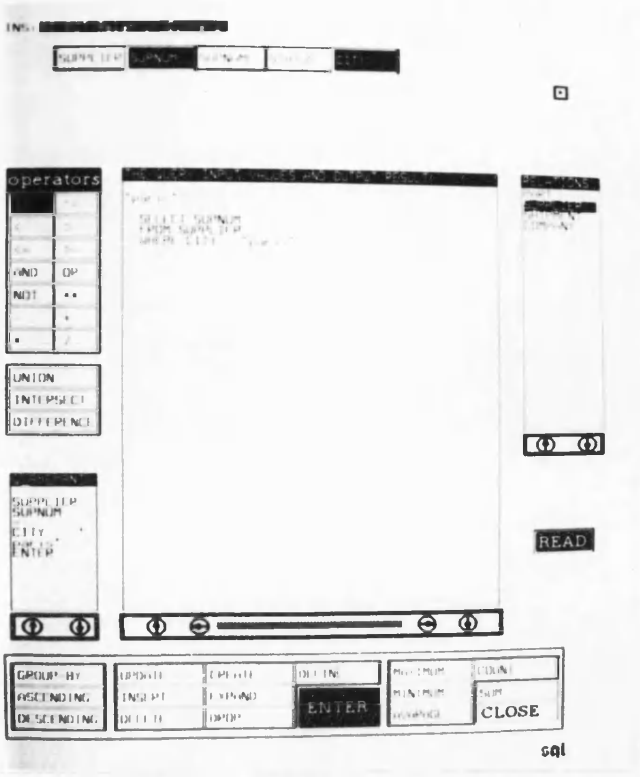
DEFINE is the command used to define an external relation from existing base relations. This relation is called view. View represents a window on the actual data, through which users can vision a particular part of the database. Operations against views are converted into operations on the real data. So that update operations on view could be dangerous, and might disturb the structure of the base relation. Therefore, these operations must be avoided on the view. SQL view's cannot be a genuine projection (with duplicate elimination), nor a join, nor a union. It also cannot involve GROUP-BY, and computed field (DATE-81).

Q26: the following example shows, the view definition in PDBIS (photo SQL13).

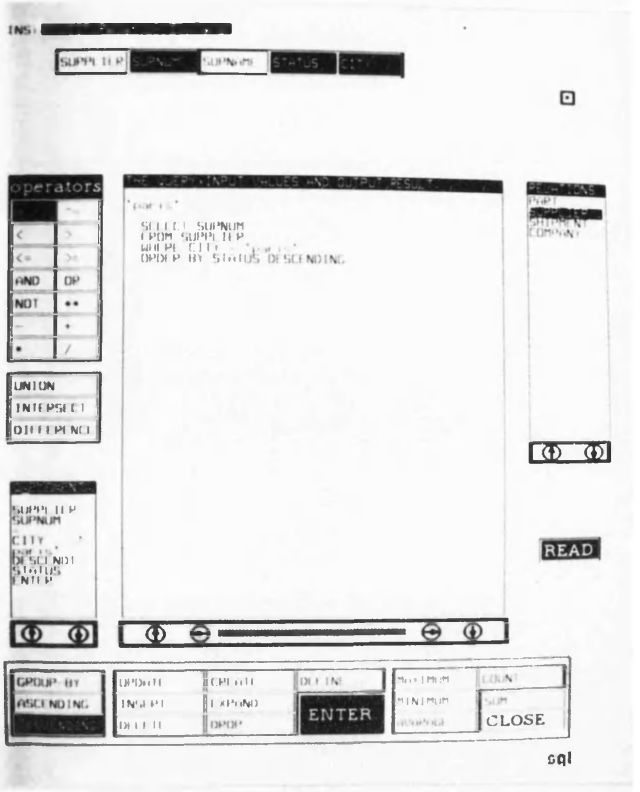
```
--------> DEFINE
--------> PART
--------> PARTNUM          DEFINE VIEW VEIW1 (VPARTNUM, VPNAME, VCITY)
--------> PNAME            AS SELECT PARTNUM,PNAME,CITY
--------> CITY             FROM PART
--------> =                WHERE PARTNUM='P2'
--------> PARTNUM
..... 'P2'
--------> ENTER

ENTER VIEW NAME: VEIW1
----------------------
```

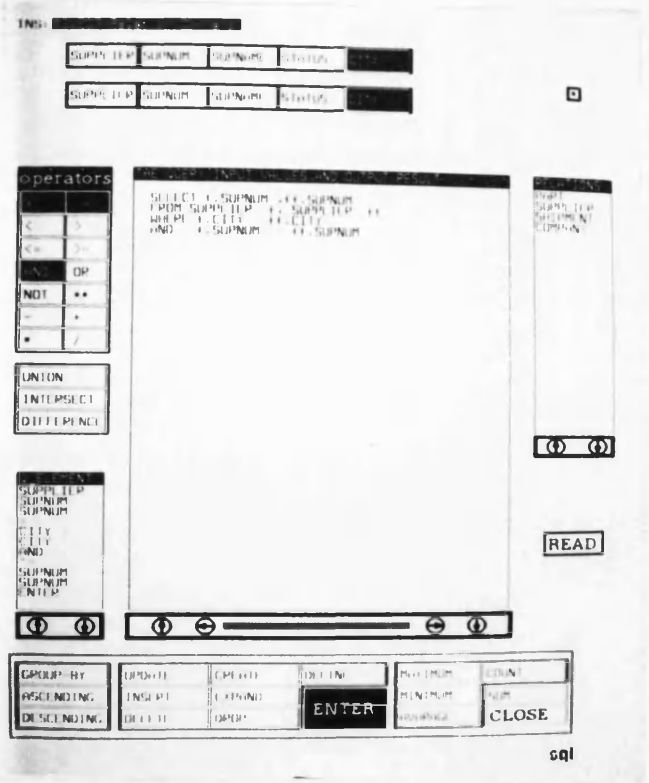
All the previous given examples are taken from DATE-81.



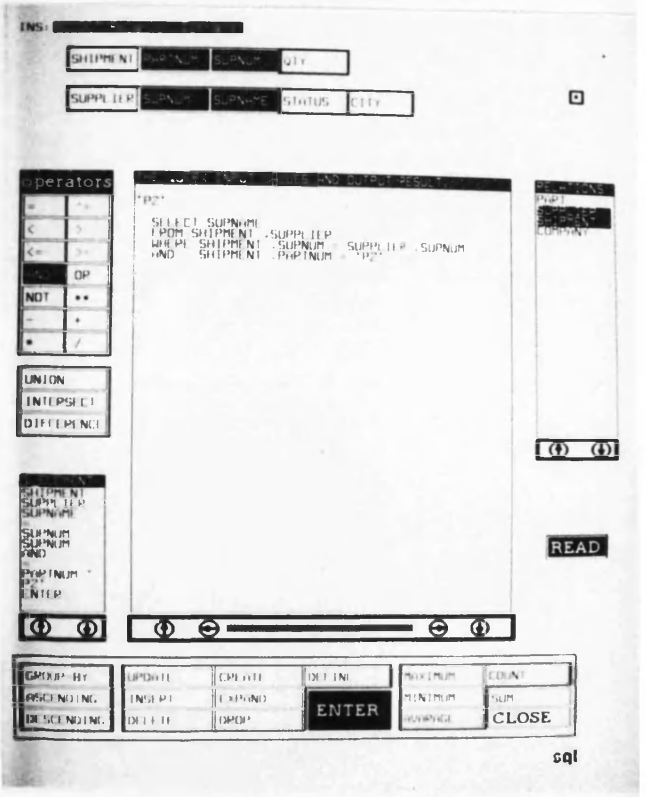
SQL1



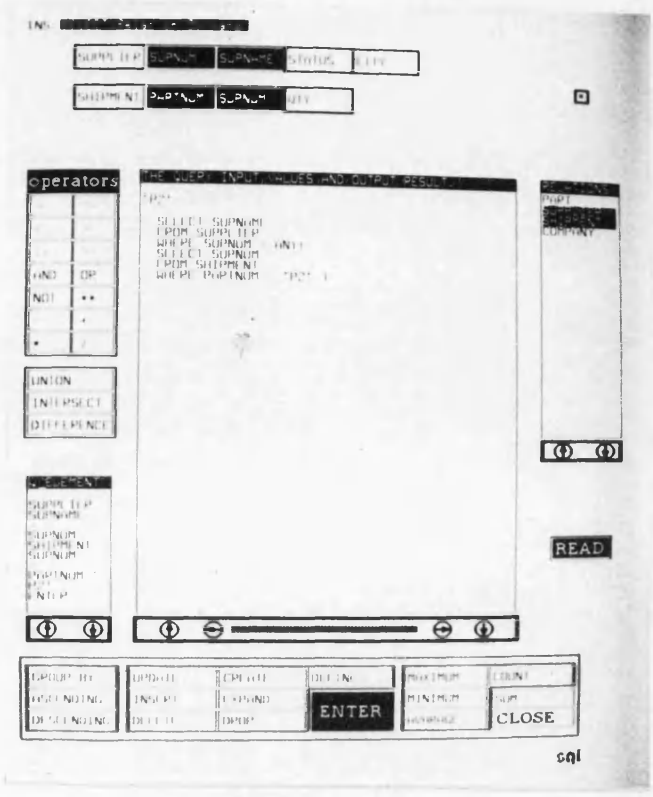
SQL2



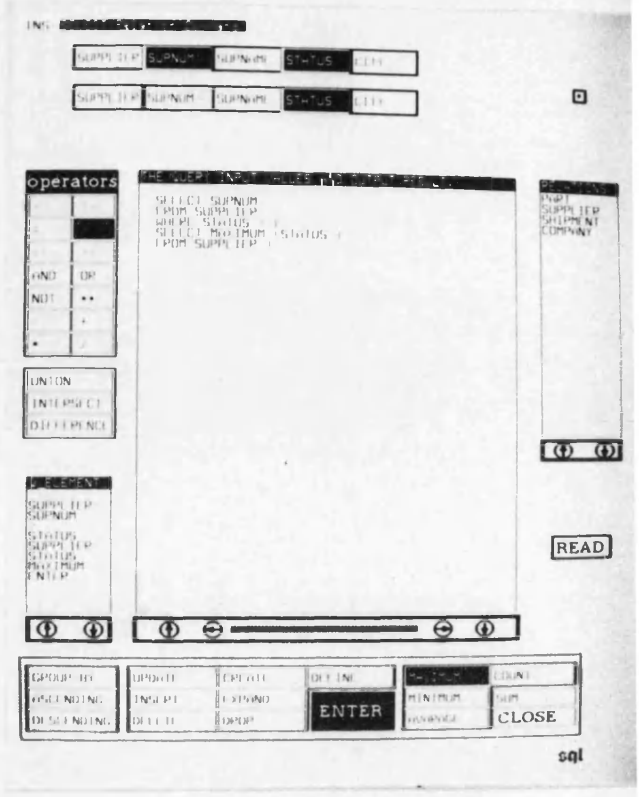
SQL3



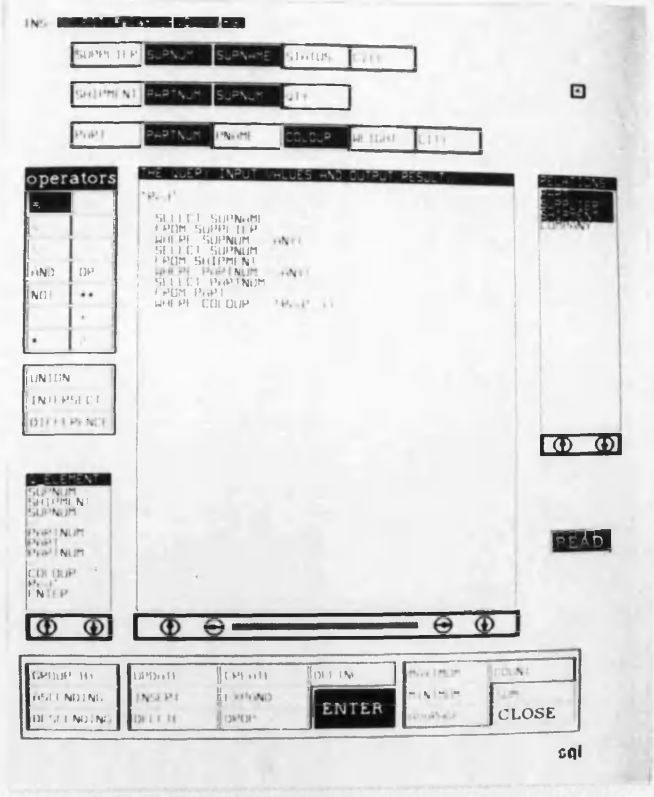
SQL4



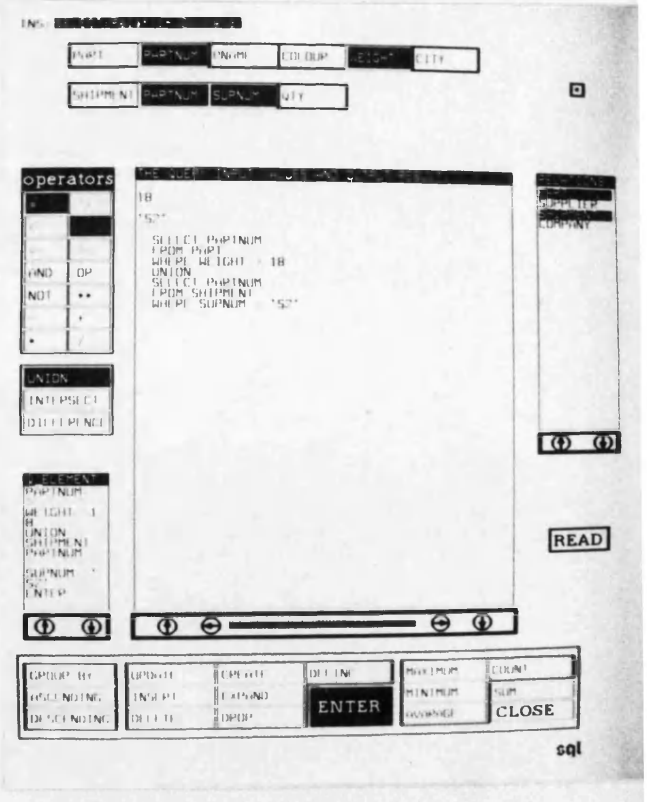
SQL5



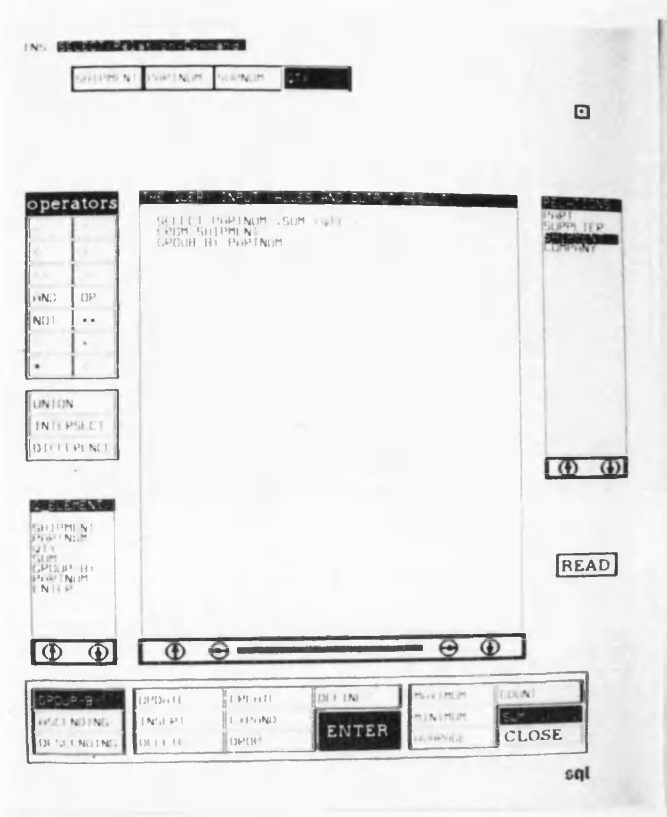
SQL6



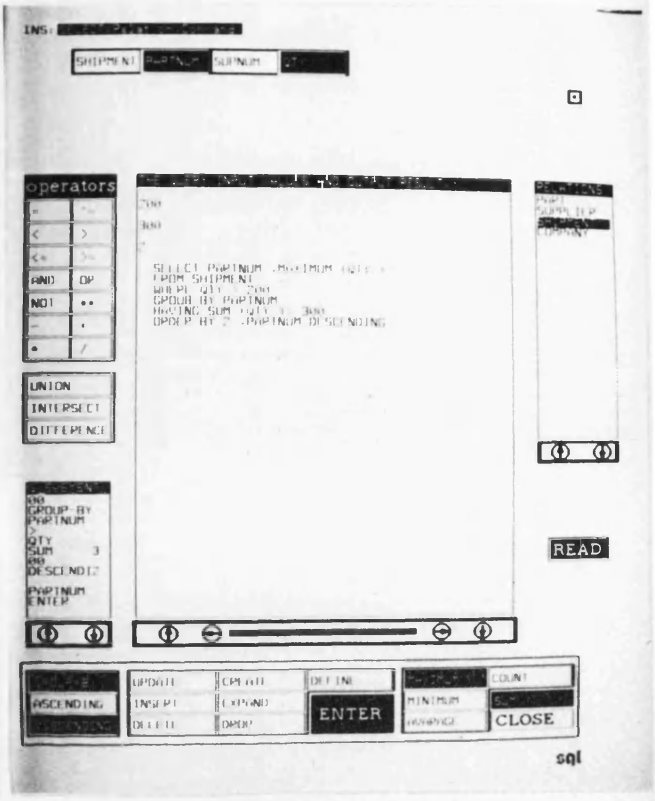
SQL7



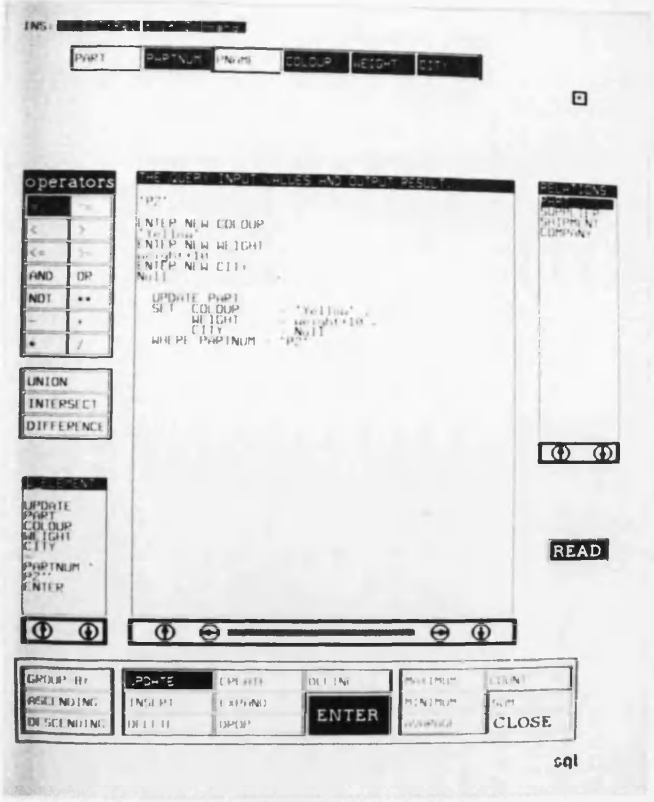
SQL8



SQL9

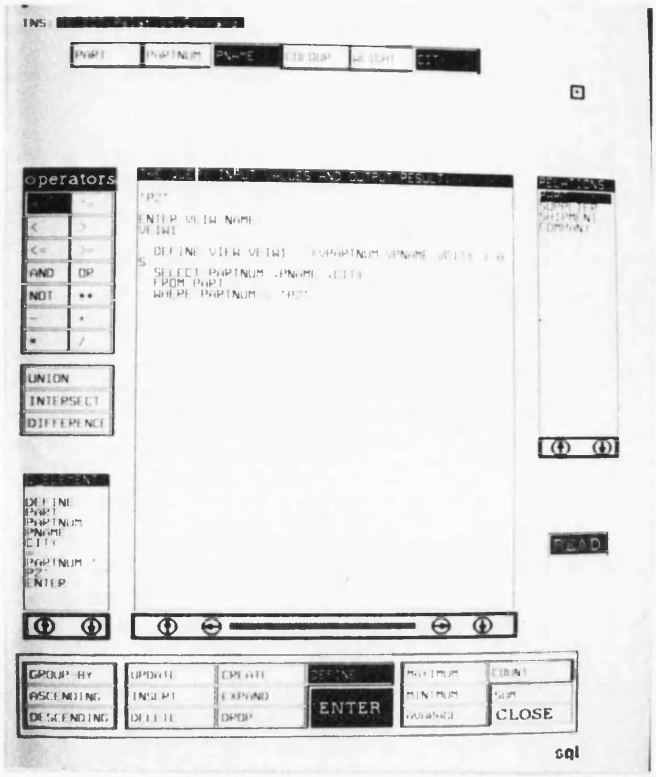


SQL10



SQL11

SQL12


operators
UNION
INTERSECT
DIFFERENCE
READ
GROUP BY
ASCENDING
DESCENDING
UPDATE
INSERT
DELETE
CREATE
EXPAND
DROP
ENTER
MAXIMUM
MINIMUM
AVERAGE
COUNT
SUM
CLOSE
sql

SQL13

CHAPTER 6

FURTHER WORK: PDBIS IN A DISTRIBUTED ENVIRONMENT

6.1. Distributed database system

Normally data in a distributed database system (DDBS) is not stored entirely at a single location, but rather is spread across a network of computers that are connected via communication links. The main objective of the distributed database system is to appear as logically centralized and physically decentralized. The users and applications access data through a single conceptual schema, but data may be physically stored in a separate computers. Thus, the user application is completely separated from the way in which the data is distributed, so that any change in data distribution structure has no effect on user applications.

The introduction of such a system into the marketplace in a large scale seems imminent. Two phenomena support this fact: the technological feasibility of DDB systems and the compatibility between the characteristics of DDB systems and the characteristics of today's database end-user interfaces. The technological feasibility is supported by the technological advancement, which has increased the performance of communications media and decreased the cost of computers and related hardware. Increasingly, database processing problems and end-user needs are driving technological solutions (GREE-81).

There are several motivational factors for adopting the distributed database technology (PACT-79), such as:

(1) Distribution of the logical database can increase security, simplify the control of database growth, and accommodate new technology.

(2) By physically placing the data where it is needed most, communication cost is reduced and response time is decreased. It also improves the database

administration by encouraging individual groups within an organization to create and maintain their local databases in order to satisfy their own requirements.

(3) Reduction in central system complexity.

(4) Reduction in the effect of central system failures.

(5) Cost advantages due to the better price/performance of mini and microcomputers.

(6) The distribution of data across several sites is essential in some environments to handle large volumes of data processing.

(7) In a heterogeneous database system (HDDBS) the user shares data, storage, and the power of different DBMSs. Generally speaking, such a system will be more available, reliable, and maintainable than the centralized system.

As we have seen the motivations for using the HDDBS exist, but the need to access relevant data is very difficult to be satisfied when the data is stored in many independent databases (SMIT-81). Formulating and implementing queries that require data from many databases, requires the knowledge of where all the data is stored, managing all the necessary interfaces, and properly combining partial results from individual queries into a single answer. These factors make the adoption of the DDB unfavourable, so that to overcome these difficulties a unified interface is needed which allows the users to access all the relevant data uniformly, while providing an automated system that performs the necessary underlying operations (LAND-82).

Implementing such an interface is very useful. For example, consider an organization that uses four different databases implemented under different DBMSs in different locations in order to provide services for different sections. To answer a question which needs data from all four databases, the user (programmer) must:

(1) Know the locations of the necessary data, the data formats, and the query languages of the local DBMSs.

(2) Decompose the question into subqueries that can be executed at each database.

(3) Arrange for the partial results to be accumulated at one database site.

(4) Write a program to extract from this accumulated data a complete response to the original question.

Without an integrated interface, this procedure is costly, time consuming, and error prone. In contrast, implementing a unified interface in a distributed environment will answer this kind of multi-database query correctly and quickly.

6.2. D-PDBIS

Many characteristics of PDBIS have met many requirements of a HDDBS. In fact, PDBIS in its current implementation represents a subset of a distributed processing system, where the query is formulated in the Perq while its execution and the data access are carried out in different machines. PDBIS provides a uniform access to remote DBMSs and presents the users with a uniform view of the data stored in different databases in the form of relations and attributes displayed on the screen. A PDBIS syntax covers a wide range of commands and functions which cause the addition of a new underlying DBMS to have no effect on its structure and mechanism. A PDBIS query does not cause any change in the local DBMS optimizations, such as access paths and database structure. Therefore, using the PDBIS as a unified interface to integrate a heterogeneous DDBMS is a natural development in its usability in data processing.

The above specifications and characteristics of PDBIS besides other factors have been encouraging enough to propose a design and specification of a prototype integrated HDDBS called 'D-PDBIS' in which PDBIS is used as a unified interface.

6.2.1. D-PDBIS overview

D-PDBIS is a software system that provides a uniform, integrated, retrieval user interface to a physically nonintegrated environment. It provides a single interface

through a single query language to data in all the databases. This system should efficiently execute queries that may require data to be retrieved from different databases that have different structures, data models, and query languages. This should be accomplished without making any change to the pre-existing databases, database management systems, or their application programs. The system will present the user with the illusion of a single, integrated, centralized database system, so that the user has to be familiar with only one uniform interface instead of numerous local system interfaces. To achieve this, the system should assume complete responsibility for knowing the locations of the local databases, accessing the data at each of the local DBMSs, resolving any data incompatibilities and combining the data to produce a single result.

To successfully execute PDBIS queries in this implementation, the system has to solve problems like: decomposing the query into a set of subqueries expressed in the different languages supported by the local DBMSs; formulating an efficient strategy for executing a sequence of subqueries and data movement steps; implementing an efficient program for accessing the data at a single local location; resolving incompatibilities between the databases, such as differences in data type; resolving inconsistencies in the same information that are stored in different databases; building the global schema; executing the original query and getting the final single answer.

Before local databases can be accessed through D-PDBIS, the local host systems must be connected to a communications network. This network can be local or geographically distributed. D-PDBIS is then connected to the same communications network, as illustrated in Figure 6.1. After this, a global user can access data in the local databases through a single interface (D-PDBIS) using a single query language (PDBIS). Each local site maintains autonomy for local database update. Local application programs can continue using the existing local DML as before, or using

PDBIS as well.

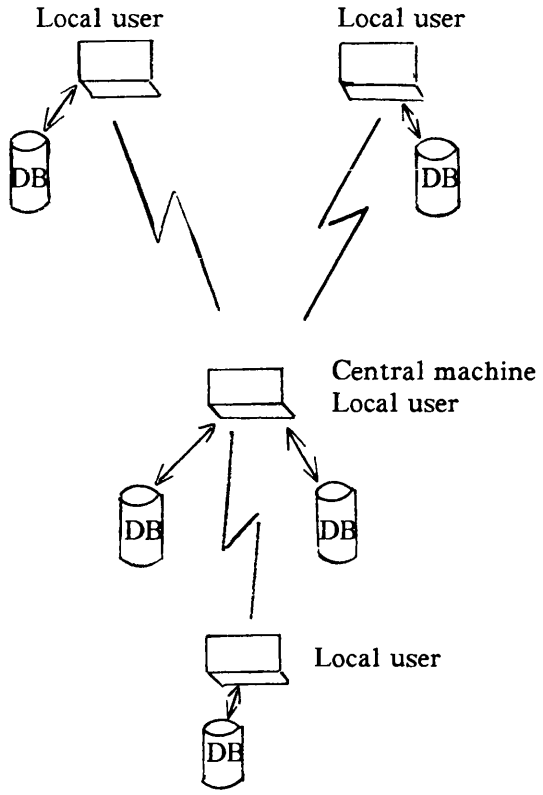


Figure 6.1 D-PDBIS network communication.

This system will not provide the user with the capability to update the data in the local database or to synchronize read operations across several locations. To have these capabilities a global concurrency control mechanism is needed, which depends on specific operations offered by the local systems (Example, locking certain local records or items). These local operations are necessary to insure consistency across the databases. Since PDBIS was originally designed to operate with a single database at a time, the tools needed to insure data consistency across databases are not globally available (not provided by PDBIS). Thus, database update is maintained locally, and the user is offered the same level of data consistency that the local DBMS provides. This gives higher database security, database integrity, and database control.

6.2.2. Design objectives

D-PDBIS design emphasises on certain objectives like generality, extensibility, and compatibility.

In order to satisfy the generality objective the system must be designed to be a general tool, capable of providing integrated access to various database systems used for various applications. Originally PDBIS had not been designed to be an interface for a specific application area.

The extensibility of the system allows expansion of functionality without major modifications, and the system must be relatively easy to couple a new local system into an existing D-PDBIS configuration.

Compatibility lets the system co-exist and be compatible with existing database systems and applications. No changes or modifications to local databases, DBMSs, or application programs are necessary in order to interface D-PDBIS with systems already in operation.

The implementation of D-PDBIS will provide the feasibility study of using PDBIS not only as a stand-alone graphical interface, but also as a backbone in other data processing systems, which particularly can be used in office automation.

6.2.3. System architecture

The major aim of the system architecture which is shown in Figure 6.1, is to provide the user with an integrated, coherent, global view of the data stored in the local databases. The user of the global database has to be insulated from the location of the data and the local system details like the data model, data description, and access language. This approach will be achieved by providing a global schema generated from the local databases for each global query. This global homogeneous schema is created in one of the sites by its local DBMS. The computer in this site acts

as a central machine which runs a central DBMS and contains the global schema for a global query. In this architecture which is shown in Figure 6.2, the local schemata are the original, pre-existing schemata defined in the local data models and used by the local DBMSs.

The global schema is an integrated view of the data that concerns a particular global query in the underlying, nonintegrated DBMSs. Therefore, for each global query there will be a relational global schema created by the central manager in the central machine. In other words the global schema stores the data in a homogeneous form to be accessed by the global query. So that, to the global user, the schema appears as if it were the schema for an integrated DBMS. In issuing PDBIS global queries against the global schema, the user need not to be aware of the location of the local DBMSs. Thus, D-PDBIS presents the user with the illusion of a homogeneous and integrated database, without requiring that the local databases be physically integrated.

The global schema will be destroyed after the execution of the global query takes place, releasing the occupied storage for the next global schema (next global query).

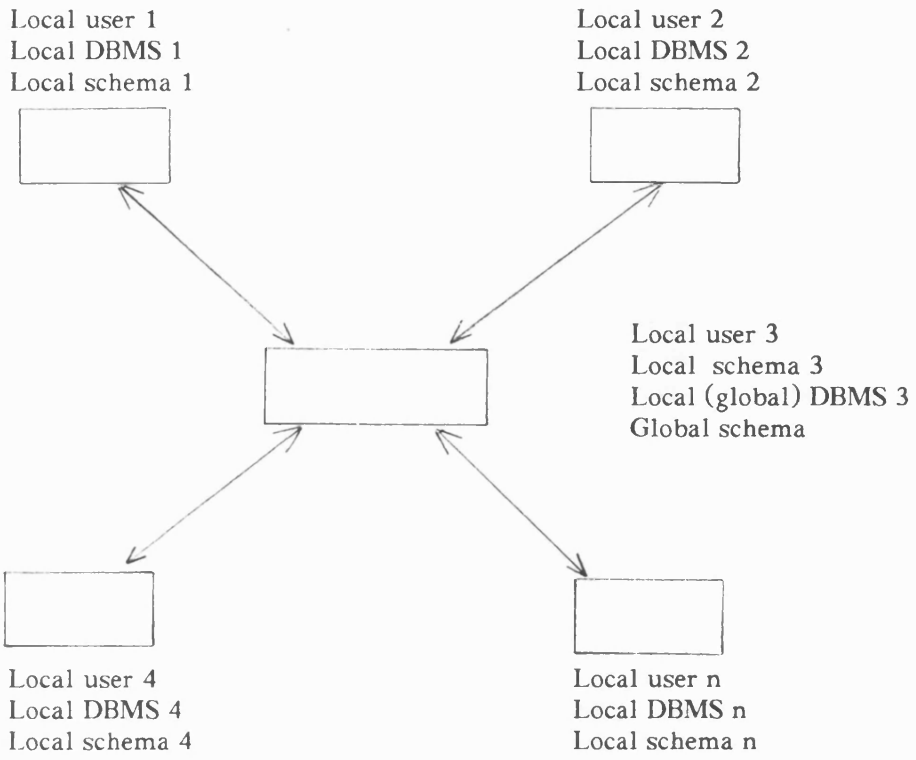


Figure 6.2 Schema architecture.

The central machine is preferred to be the one which has the large storage capacity and the most complete and powerful DBMS. This will compensate for any shortage of storage and lack of power in the local database management systems, because the original global query is executed by the central DBMS and data is accessed from the global schema.

The central machine will be responsible for managing the communications among the different sites in the network. This can be done by applying a communication protocol, coupled with a piece of software which controls the flow of information in the central machine as well as between the central machine and the rest of the network. Other software in the central machine is required to create the global schema and store the related information.

6.2.4. Components architecture

The components architecture of D-PDBIS is shown in Figure 6.3. There are two type of modules: a central manager and a local controller. All global aspects of a query are handled by the central manager. These aspects include the formulation of the subqueries to retrieve the relation or the relation equivalent (as in the CODASYL model) from the local systems, managing the communications between the central machine and the local sites, creating the global schema, and executing the global query. There is one local controller for each machine. The main task of the controller is to transmit the data values of the selected relation from the local machine to the central data manager once it is retrieved.

When a query includes multiple relations stored in different databases, these relations have to be brought to the central machine in order to formulate the global schema before the global query is processed. Therefore, it is necessary to decompose the global query into subqueries, so that each subquery accesses a single database, and then a copy of each selected relation is sent back to the central machine. In D-PDBIS, the decomposition techniques are part of the query formulation process. The relations which are involved in satisfying the query result and are necessary for the query construction are actually accessed from the different databases during the query formulation process, and by the time that the query formulation is completed the global schema is created at the central machine.

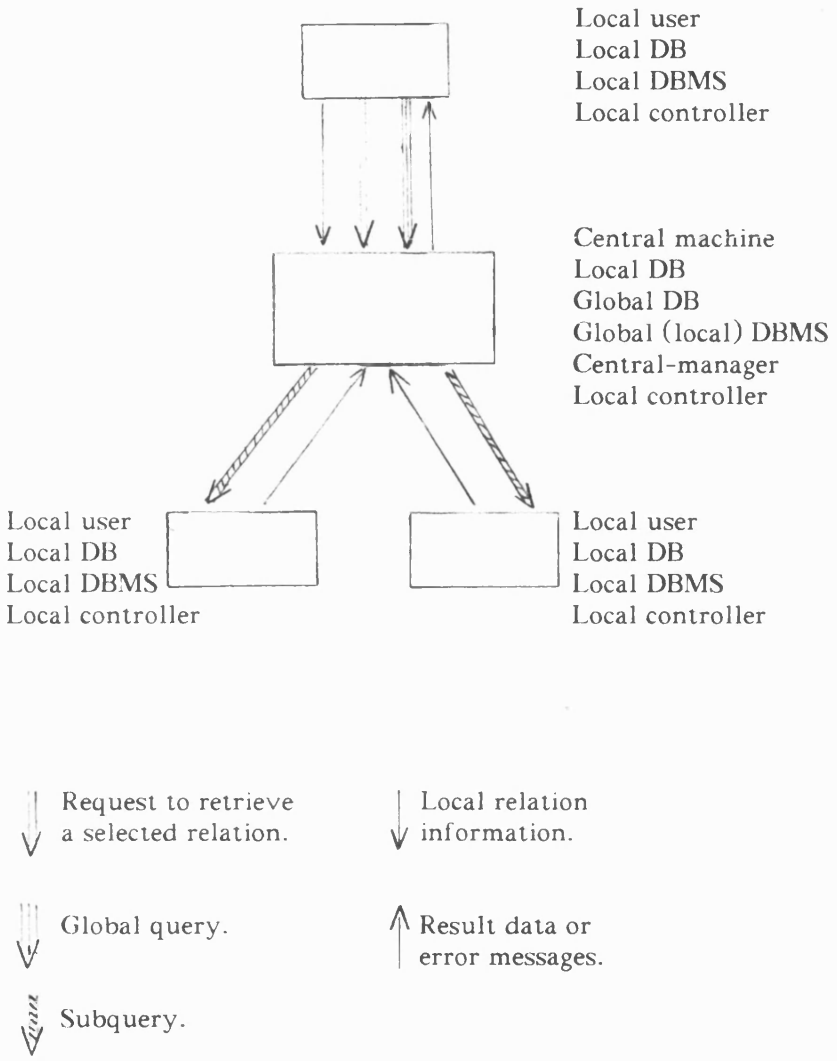


Figure 6.3 D-PDBIS components architecture and information flow.

In D-PDBIS interface, the relations menu on the user screen should contains the relation names of all the databases in the network instead of the relation names of a single specified database as in PDBIS implementation. There will be two underlying directories, the first one resides at the user's local machine which contains the names of the databases and their relations in the local machine. The second directory resides on the central machine and contains the name and location of the database for each relation displayed on the relations menu in the local machines. These directories are

maintained by a program automatically executed at the local machine whenever the database at that location is changed. Having two separate directories will reduce the communication overhead, software, and processing.

6.2.5. The query processing

The user formulates his query on his local terminal by selecting relations, attributes, functions and operators. The selected relation is first examined against the local directory to find out whether it belongs to the local database or not. In the first case, the relation is directly retrieved and its structure is displayed on the user's screen, while its data values are transmitted to the central machine where the central-manager software removes the unnecessary information and solves the incompatibilities between the relation structure and the structure defined by the central DBMS. Then the relation is defined and stored in the global schema. In the second case when the relation belongs to another local database, it is retrieved according to a request transmitted to the central manager by the local controller. The central manager finds out the location of the relation by examining the central directory, and then constructs and transmits the necessary subquery to the local database where it will be executed and the result will be sent back to the central manager. Then the incompatibility is solved and the data are refined in order to define and store the relation in the global schema. In the meantime, the relation structure is sent back to the user local machine by the central manager to be displayed in its specified skeleton. Figure 6.3 shows the process and the information flow among D-PDBIS components.

It might be not sensible to transmit data to the central machine when the query can be handled locally. In fact there are two techniques which can be used to process the query, these have a number of trade-offs between them. The first technique is to keep the local relation data values in the local machine until the end of the query

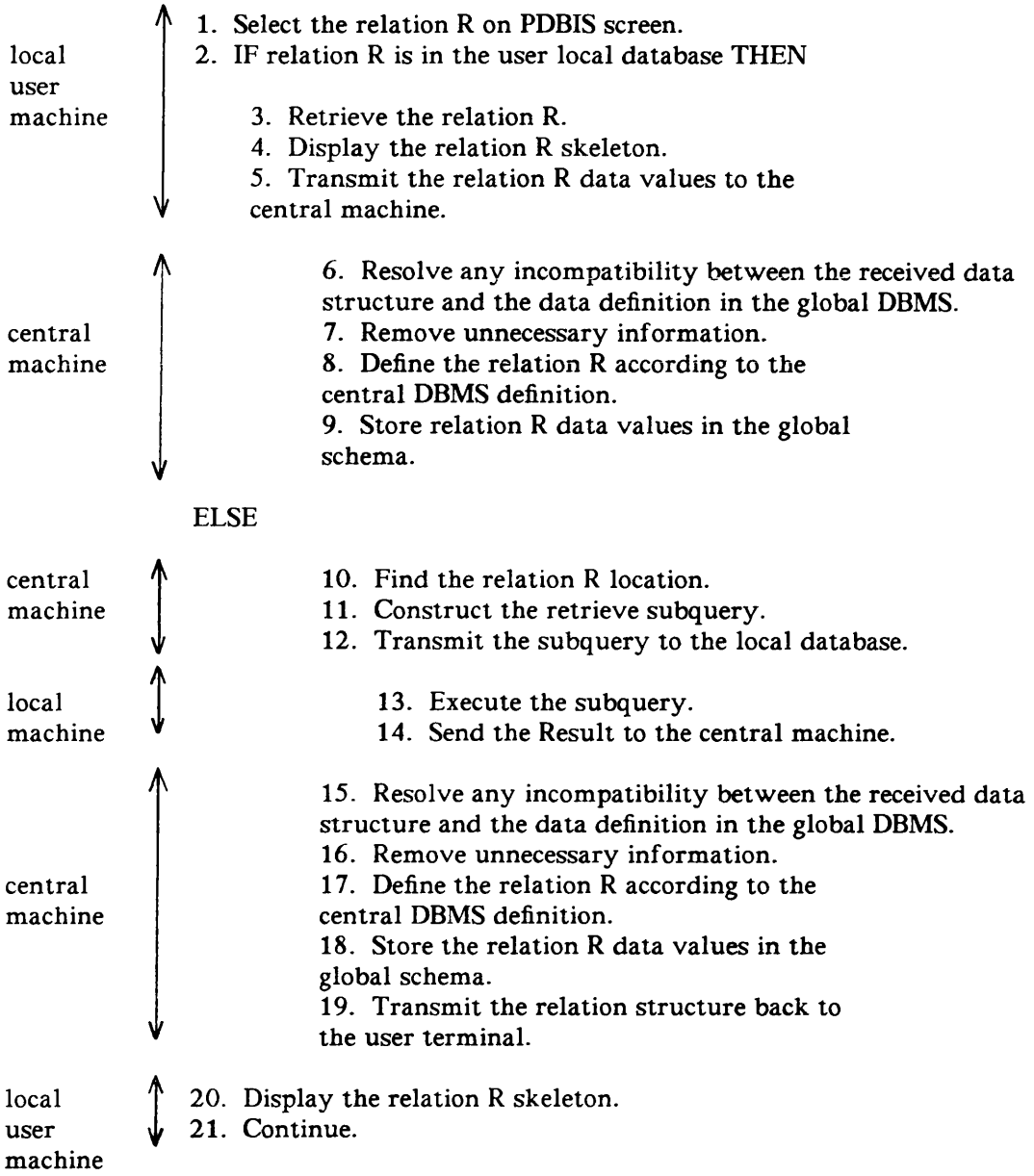
formulation process. Then the controller transmits the local data values to the central machine only when the query is global, otherwise the controller prepares for the execution of the query locally. In this case the creation of the global schema is completed after the formulation of the global query. In this technique the power of the global DBMS is not available to the local queries, therefore some of the local queries which can not be handled locally by the local DBMS will be rejected.

The second technique is to treat everything globally. As it is explained earlier, the values of any selected relation whether it is a global or local relation is transmitted to the central machine and stored in the global schema immediately after it has been retrieved from the database. In this technique, when the query is global the whole process might gain some speed, but when the query is local the whole process needs more time to be completed. On the other hand the power of the global DBMS is available to the local and the global queries.

However, by having two modes of PDBIS to use, the global mode, and the local mode, the user can select the mode which is more appropriate for his particular query. For instance if the user knows that all the selected relations exist in the local database, he should select the local mode and use PDBIS in its traditional way as an interface for one central DBMS.

The processing of a global query through D-PDBIS components can best be illustrated by the following algorithm.

Suppose that the relation R is to be selected through the formulation process of the query.



After the global query formulation is completed, it will be transmitted to the central machine in the form of the central DBMS language, to be executed against the global schema.

According to the hardware equipments and the software utilities that are available here at the University computer laboratory, and also according to the amount of time available for this project, it has not been possible to implement such a system at the present time, but it can be a good topic for future research.

CHAPTER 7

CONCLUSION

This dissertation has described the design and the implementation of a graphical database interface system, PDBIS. In this chapter the highlights of this work will be summarised.

7.1. End User Interface

The end user interface has been defined in this thesis as a facility which enables the non-computer professional to access information stored in computerized files. The self contained high level data languages which enable the users to query the database are called query languages. The query language has been found more suitable to be used with the relational data model than with the hierarchical and the CODASYL models. Most of the existing query languages are intended to be easy to use and easy to learn by the casual users, those who interact with the system occasionally and usually they do not have a background in computer. It has also been found that both the linear high level query language and the natural query language in its current state of development cannot be the easy to use and easy to learn language which satisfies the requirements of all classes of end user. Therefore attention has been directed toward the two dimensional graphical specification. Unfortunately none of the resulting languages has achieved the desired specification of an easy-to-use and easy-to-learn language and at the same time satisfy the requirements of the entire spectrum of end users. Each of them have problems ranging from navigation through database structure to the lack of feedback and query construction complication. Consequently my system PDBIS, has been designed to overcome these problems and adds other approaches and features to achieve the desired interface, taking in the consideration the recent development in information technology and the increasing demands of end users.

7.2. PDBIS

PDBIS has been designed not only to be unified for a number of relational database management systems operating in various machines, but also to be unified for various DBMSs supporting different data models. In fact this is the main approach of achieving the primary objective of this project, to use PDBIS in a distributed environment and in office automation.

PDBIS has managed to employ the powerful graphics in the Perq machine to present the database to the user in its most natural form, and separate the user from the physical structure of the data, the underlying DBMS, and the data manipulation language. The way in which the database is displayed on the screen allows the user not to be concerned with knowing the database structure and its relationships, and also removes the navigation process that exist in other graphical interfaces.

PDBIS syntax is complete and covers all significant functions, operators, and operations that are required in most relational DBMSs. This syntax is well suited to the expression of a wide range of queries against a database, and this is especially true as queries become more complex. This style of syntax allows PDBIS to be easily expanded by accepting any new DBMS to operate underneath, simply by adding a new analyzer to the system. The PDBIS primitives are laid out on the screen in a most efficient form. The design of this layout was based upon many considerations categorized as user, hardware, software, and application consideration. Therefore, the problems of screen size, database size, and user interaction, which exist in the previously mentioned graphical systems are avoided here. The scrolling facility overcome the screen size limitation and the size of the database. Having PDBIS layout, the data structure, and a number of styles of feedback make the query construction process very easy, especially with using the efficient selection device, the Puck, leaving the typing problems or the complication of using the Light Pen behind. These

specifications of PDBIS design have great effect on speeding the computer processing, reducing the incidental user errors, and increasing user productivity.

Providing three different styles of feedback have successfully provided all the expected feedback advantages to all classes of PDBIS user. By having more than one style of feedback, any negative aspect which might appear in one style, will be covered by the other, so that at the end the three different styles act as one successful integrated feedback method. This technique of feedback cannot be found in other comparable graphical interfaces like QBE, FORAL-LP, and CUPID.

As PDBIS is intended to be unified for a number of different DBMSs, the chance of using the interface by different classes of user is very high. Therefore PDBIS has been designed to explain itself to the user. The way in which the database is presented to the user and instruction statements provided very much minimizes, if not removes, the amount of training required to get the user started querying the database. The query formulation process is very easy, providing that the user knows what he wants and understands the logic of the query. This is due to the menu selection style; the simple and accurate selection device; and the flexibility in selection order of the query primitives. The dialogue style in the updating queries ensure the user requirement and lead him to formulate the query smoothly and correctly, avoiding the disturbance of the database that can be caused by incorrect updating queries. In many queries the user does not have to use the keyboard at all. Only constant values has to be entered through the keyboard, and this is found to be much faster than the selection of each character of the constant value from the screen, as in FORAL-LP.

In addition to the above characteristics of PDBIS, the degree of portability in it is high due to several factors; the simplicity of the PDBIS program structures allows PDBIS to be implemented on a relatively small size memory machine; the independence of PDBIS from the Perq operating system and the underlying DBMS permits PDBIS to

be run in any other machine with similar graphics facility to the Perq, that is a high resolution bit mapped screen and pointing device; using Fortran-77 as an implementation language makes for easy maintenance, development, and transformation of the system, even to the unsophisticated user.

PDBIS is one of the very few database interface designs which provides the facility for report generation, graph display, and to have the result of the query printed in different output formats. By this utility the user can control the output of his query and specify the format which is most suitable for that particular requirement. Therefore the database efficiency is increased by better utilizing the accessed data. This utility makes PDBIS into a multifunctional interface which allows the user to perform several different functions using the same utility. This certainly increases the productivity and positively effects the cost. Furthermore, this utility demonstrates the ability of using PDBIS with other data processing functions, and will support its integration with office information system.

Using PDBIS in distributed environment as explained in chapter 6 gives an idea of how PDBIS can be expanded from its primary design as a stand alone easy to use and easy to learn end user interface, to be an important component around which an integrated information system can be built.

The further work proposed in this thesis was given as a complete design and details explanation of the important elements of that work, rather than just outlines. This is has been done in order to give the clear idea of the proposed work and encourage other people who have the interest to continue the work by developing this design and so achieve the strategic objective of having an easy to use, easy to learn, unified end user interface used in a variety of data processing environments.

The successful representation and manipulation of information graphically in systems such as PDBIS and the other recent developed graphical information systems,

will provide incentive for further graphics development. As these systems become more popular than the keyboard interfaces, better and less expensive graphics hardware and software will be developed.

In conclusion, this work may be considered a feasibility study and a pilot implementation of an optimal graphical end user interface. As such it has been shown that such an interface can be designed and implemented in a reasonable amount of time and effort. It remains for future research to pursue the proposed work in this thesis in order to achieve the remaining objectives of the interface.

APPENDIX A

CODD INTERFACE

1. SALT analyzer

As we have mentioned earlier SALT is a simple language based on relational algebra, and used in both batch and interactive mode to access database managed by the CODD database management system (KING-79). SALT does not provide the casual user with the full utility to retrieve and update the database, it is more efficient with simple retrieval queries. One important feature of SALT is the definition of variables to take the value of a relational expression. SALT in this respect is similar to ISBL in PRTV (TODD-76). Expressions are constructed using operators to indicate various relational operations, and variables or stored base relations as operands. This approach allows retrieval requests to be made in a simple fashion, saving processing time and spaces, as the variables are treated externally from the database. This also provides the user with more flexibility in defining the database structure, and permits the database administrator to keep only the necessary information in the database.

The SALT language does not support updating operations or other functions like grouping, or ordering the output result, neither does it provide any arithmetic operation or it have any built-in function. Therefore, SALT analyzer is not as comprehensive as SQL analyzer. In fact, it supports only a subset of SQL analyzer syntax, which covers only the retrieval operations which may consist of various relational operations and may involve a general structure of the conditional clauses, with logical operators.

In SALT analyzer, the analysis process is less dependent on the construction of the mapping clauses in the same fashion as in the SQL analyzer. That is because the SALT language is less structured than the SQL language. The analysis target here is to find out how many join operations are involved in the query, restructuring the base

relations which do not have the join key in the first columns, by defining variables. This is necessary as SALT joins always occurs over the first columns of the two relations. These defined variables express simpler relations, which consist only of the attributes which are necessary to answer the query, and contains the join key in their first columns. SALT statements are then constructed by specifying the relational operations on these variables or relations.

2. PDBIS-SALT Interface

The current implementation of PDBIS-SALT interface is presented in this section through a number of examples which run against a database controlled by CODD database management system, operating on the DARKSTAR 68000 minicomputer.

In the following PDBIS representation the arrows indicate the selected elements of the query on Perq screen, while the dotted lines specify the typed constant values.

Q1: Get full details for all suppliers (photo SALT1).

PDBIS representation	SALT representation
--------> SUPPLIER	
--------> *	LIST SUPPLIER
--------> ENTER	

This query project all the attributes in the relation SUPPLIER.

Q2: Get part numbers for all parts supplied.

--------> SHIPMENT	
--------> PARTNUM	LIST SHIPMENT % PARTNUM
--------> ENTER	

The '%' is the projection operator in SALT language. In the above query PARTNUM is projected out of the relation SHIPMENT.

Q3: Get supplier numbers for all suppliers in Paris.

```
--------> SUPPLIER
--------> SUPNUM          a1 = SUPPLIER : CITY = "Paris"
--------> =               LIST a1 % SUPNUM
--------> CITY
..... "Paris"
--------> ENTER
```

The above query includes a condition clause to be satisfied before the projection of SUPNUM. In SALT, the condition clause is indicated by the operator ':'. a1 is the variable which hold all the tuples of the relation SUPPLIER which have CITY = Paris.

Q4: Get the supplier numbers and the supplier names in London with status > 10 (photo SALT2).

```
--------> SUPPLIER
--------> SUPNUM
--------> SUPNAME
--------> >
--------> STATUS          a1 = SUPPLIER : STATUS > 10
..... 10                  & CITY = "London"
--------> AND             LIST a1 % SUPNUM, SUPNAME
--------> =
--------> CITY
..... "London"
--------> ENTER
```

The query has introduced the use of the logical operator '&' to link between two conditions.

Q5: Get part numbers for parts that either weigh more than 10 pounds or are currently supplied by supplier S2 (or both) (photo SALT3).

```
--------> PART
--------> PARTNUM
--------> >
--------> WEIGHT
..... 10                    a1 = PART : WEIGHT > 10
--------> UNION             a2 = SHIPMENT : SUPNUM = "S2"
--------> SHIPMENT          f = a1 % PARTNUM
--------> PARTNUM           g = a2 % PARTNUM
--------> =                 LIST f + g
--------> SUPNUM
..... "S2"
--------> ENTER
```

This query shows the use of the relational operator UNION (+), between two sets of data values produced by two subqueries, and assigned to two variables, 'f' and 'g'.

Q6: For each part supplied get part number and names for all cities supplying the part (photo SALT4).

```
--------> SUPPLIER
--------> SHIPMENT
--------> PARTNUM           b = SHIPMENT % SUPNUM, PARTNUM
--------> CITY              d1 = (SUPPLIER * b)
--------> =                 LIST d1 % PARTNUM, CITY
--------> SUPNUM (in SUPPLIER)
--------> SUPNUM (in SHIPMENT)
--------> ENTER
```

In the above query a join operation has been done between relation SHIPMENT and SUPPLIER, in order to project PARTNUM and CITY. The variable 'b' is defined to express the relation SHIPMENT with the join key SUPNUM in the first column. The query also shows that the join condition does not appear in the SALT representation. The join in SALT is regarded by default as equijoin over the element in the first columns.

Q7: Get supplier names for suppliers who supply part P2 (photo SALT5).

```
--------> SHIPMENT
--------> SUPPLIER
--------> SNAME
--------> =                          a= SHIPMENT % SUPNUM, PARTNUM
--------> SUPNUM (in SHIPMENT)      d1=(SUPPLIER * a) : PARTNUM = "P2"
--------> SUPNUM (in SUPPLIER)      LIST d1 % SNAME
--------> AND
--------> =
--------> PARTNUM
..... "P2"
--------> ENTER
```

The above query shows a retrieving condition applied to a relation resulted from the join between relation SHIPMENT and SUPPLIER.

Q8: Get part numbers and quantities of all parts supplied from london.

```
--------> PART
--------> SHIPMENT
--------> PARTNUM
--------> QTY                        d1 = (PART * SHIPMENT) : CITY = "London"
--------> =                          LIST d1 % PARTNUM, QTY
--------> PARTNUM (in SHIPMENT)
--------> PARTNUM (in PART)
--------> AND
--------> =
--------> CITY
..... "London"
--------> ENTER
```

In the above query none of the joined relations is assigned to variables, because they contain the join key in their first columns. All the tuples of the resulted relation from the join operation, which have CITY = London are assigned to the variable d1 from which PARTNUM and QTY are projected.

Q9: Get part numbers and quantities for parts that either of colour red or are currently supplied by supplier S2 (or both).

```
--------> SHIPMENT
--------> PART
--------> PARTNUM
--------> QTY                          a2 = SHIPMENT : SUPNUM = "S2"
--------> =                            d1 = (PART * SHIPMENT):COLOUR="Red"
--------> PARTNUM (in SHIPMENT)        f = d1 % PARTNUM, QTY
--------> PARTNUM (in PART)           g = a2 % PARTNUM, QTY
--------> AND                         LIST f + g
--------> =
--------> COLOUR
..... "Red"
--------> UNION
--------> SHIPMENT
--------> PARTNUM
--------> QTY
--------> =
--------> SUPNUM
..... "S2"
--------> ENTER
```

The above query involved the use of relational operator between two relations, the first is produced by a join operation between SHIPMENT and PART under the projection condition COLOUR = Red, while the second produced as a result of project operation on relation SHIPMENT under the condition SUPNUM = S2.

Q10: Get the number of parts with colour green supplied by all suppliers (photo SALT6).

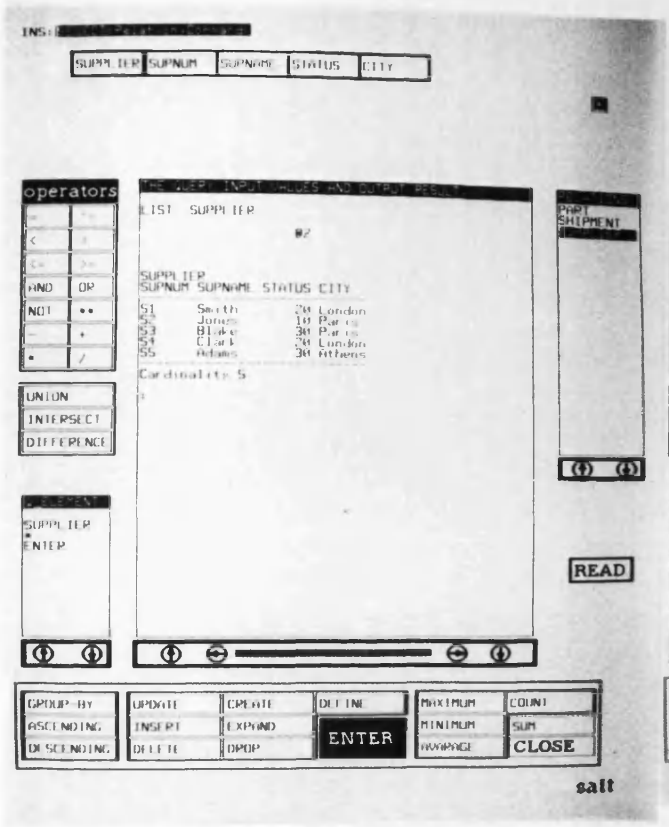
```
--------> PART
--------> SHIPMENT
--------> PARTNUM                     d1 = (PART * SHIPMENT) : COLOUR = "Green"
--------> COUNT                       COUNT d1 % PARTNUM
--------> =
--------> PARTNUM (in PART)
--------> PARTNUM (in SHIPMENT)
--------> AND
--------> =
--------> COLOUR
..... "Green"
--------> ENTER
```

SALT provides the command COUNT to count the number of tuples (records) in a relation. The above query shows how the command COUNT can be employed in PDBIS.

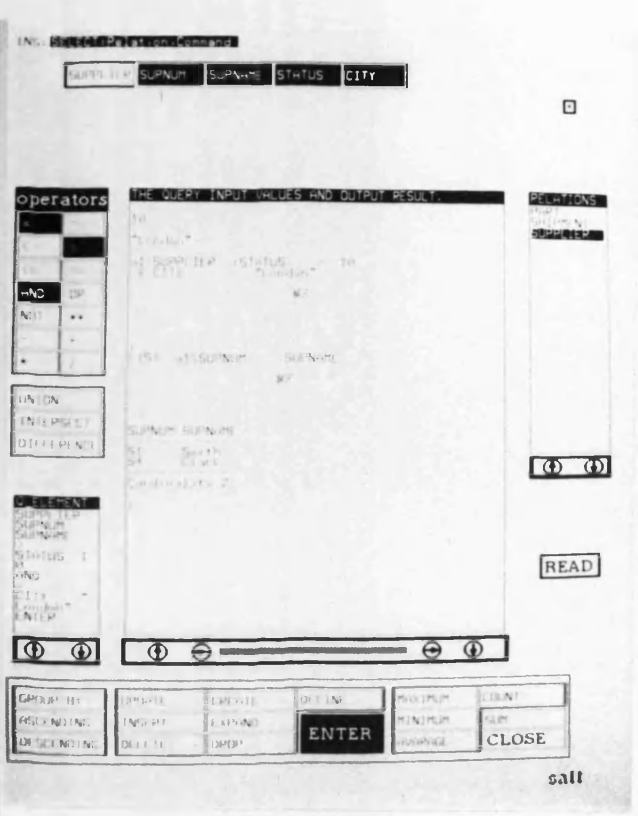
Q11: Get supplier names for suppliers who supply at least one red part (photo SALT7).

```
--------> PART
--------> SHIPMENT
--------> SUPPLIER
--------> SUPNAME
--------> =                          b = SHIPMENT % SUPNUM, PARTNUM
--------> SUPNUM (in SUPPLIER)      d = (b * SUPPLIER) % PARTNUM, SUPNAME
--------> SUPNUM (in SHIPMENT)     e = PART % PARTNUM, COLOUR
--------> AND                      f = (e * d) : COLOUR = "Red"
--------> =                        LIST f % SUPNAME
--------> PARTNUM (in PART)
--------> PARTNUM (in SHIPMENT)
--------> AND
--------> =
--------> COLOUR
..... "Red"
--------> ENTER
```

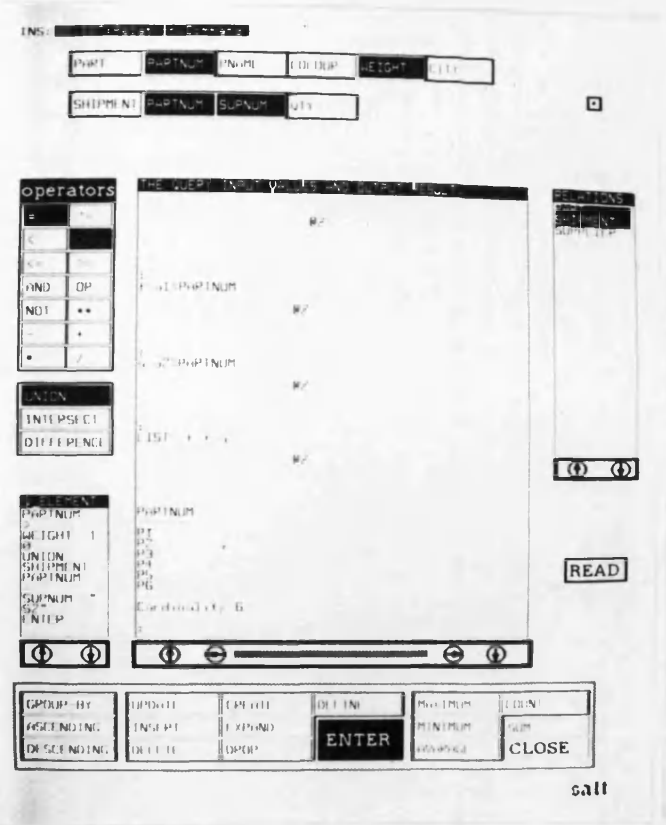
In the above query the three relations SHIPMENT, SUPPLIER, and PART are joined together to produce the right answer. Two join operations have been performed, the first was between SUPPLIER and SHIPMENT over the join key SUPNUM, while the second join was between the resulted relation from the first join and the relation PART, over the join key PARTNUM.



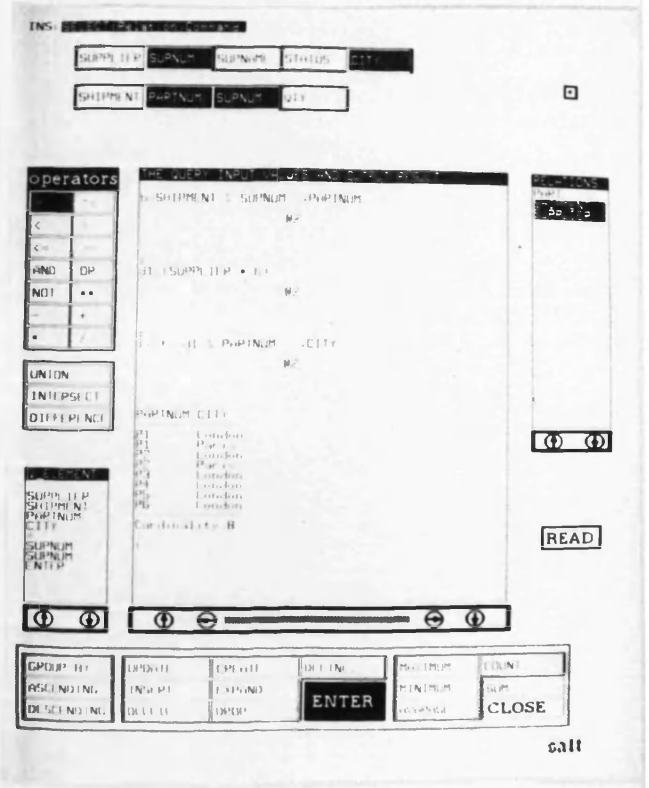
SALT1



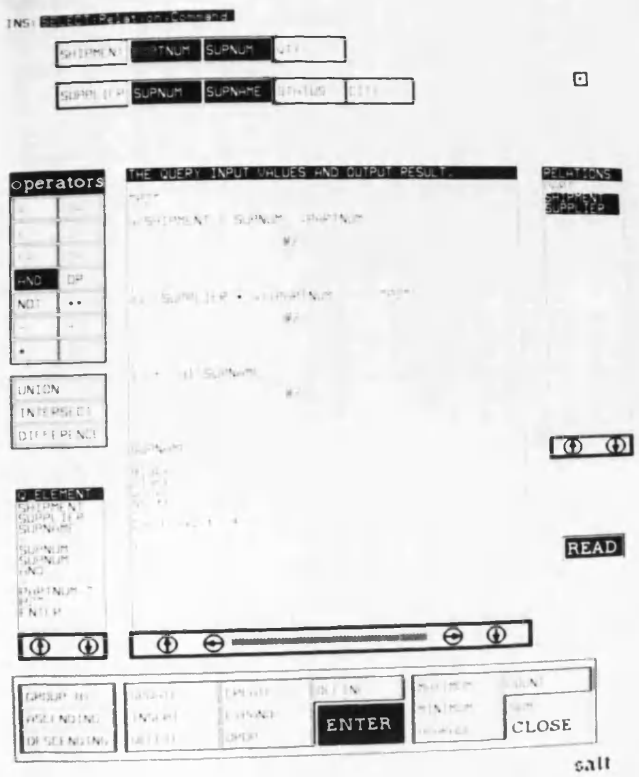
SALT2



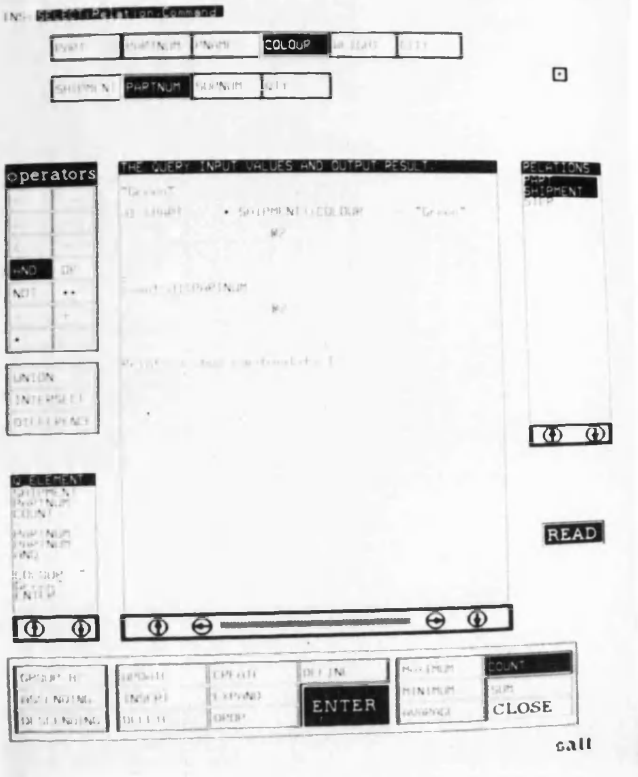
SALT3



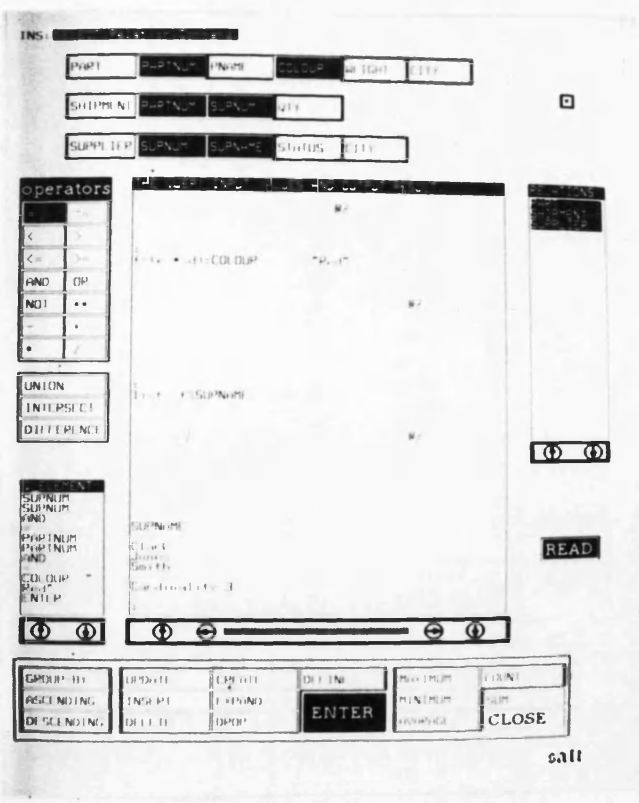
SALT4



SALT5



SALT6



SALT7

APPENDIX B

DBASE-II INTERFACE

1. DBASE-II analyzer

DBASE-II analyzer handles the analysis and the translation of PDBIS representation for retrieving and updating queries into a set of DBASE-II commands, which then, transmitted to NORTHSTAR, Z80 based minicomputer to access data stored and managed by DBASE-II (ASHT-81).

DBASE-II manipulation facility consist of a wide range of retrieving, updating and other manipulation commands, but DBASE-II analyzer implements only those commands which are essential to retrieve and update the database, and exist in most database manipulation languages. Some of the updating commands which are covered by PDBIS syntax have not been implemented in the current PDBIS-DBASE-II interface implementation, due to the dialogue nature of these commands in DBASE-II, which require a different communication protocol from the one currently used.

The syntax formats of DBASE-II analyzer represents a subset of the syntax formats that are covered by SQL analyzer. The retrieving part of this syntax is built around the recognition of selection, join, and projection operation which might be done under certain specification represented by the general structure of the conditional clause. In this interface, up to four conditional clauses may be specified to satisfy the projection of certain attributes in one query. These conditional clauses are linked by logical operators.

The two DBASE-II commands, COUNT and SUM have been covered in DBASE-II analyzer as a two built-in functions. COUNT is used to count the number of records in a database file, or to count a certain number of records satisfying certain conditions, while SUM is used to total values of numeric fields in a database file or to total values of numeric fields satisfying certain condition in a database file.

Arithmetic operations as well as the ORDER-BY clause are covered by this analyzer to be used in the same formats which are presented in the SQL analyzer (see chapter 5).

2. PDBIS-DBASE-II Interface

Beside the retrieving operations the current implementation of PDBIS-DBASE-II interface covers updating operations like UPDATE, DELETE, DROP, and DEFINE. Other updating operations like INSERT, EXPAND, and CREATE are not implemented in the current version due to the previous mentioned reasons.

In this section the characteristics of the interface is introduced through a number of examples, supported by the necessary explanation.

2.1. Retrieving operations

In the following PDBIS representation the arrows indicate the selected elements of the query on the Perq screen, and the dotted lines specify the typed constant values.

Q1: Get full details for all suppliers.

PDBIS representation	DBASE-II representation
--------> SUPPLIER	USE SUPPLIER
--------> *	LIST
--------> ENTER	

In above example, LIST is used to project all the components of the current relation (database file in DBASE-II terms) SUPPLIER.

Q2: Get part numbers and quantities for all parts supplied.

--------> SHIPMENT	
--------> PARTNUM	USE SHIPMENT
--------> QTY	LIST PARTNUM, QTY
--------> ENTER	

The above query projects only part of the relation SHIPMENT.

Q3: Get supplier numbers for all suppliers in London (photo DB1).

```
--------> SUPPLIER          USE SUPPLIER
--------> SUPNUM            LIST SUPNUM FOR CITY = 'London'
--------> =
--------> CITY
..... 'London'
--------> ENTER
```

The above query shows the use of the conditional clause identified by 'FOR'.

Q4: Get supplier numbers for suppliers in Paris with status < 50.

```
--------> SUPPLIER          USE SUPPLIER
--------> SUPNUM            LIST SUPNUM FOR CITY = 'Paris' .AND.
--------> =                 STATUS < 50
--------> CITY
..... 'Paris'
--------> AND
--------> <
--------> STATUS
..... 50
--------> ENTER
```

The above query involves two conditional clauses linked by the logical operator AND, to be satisfied before the projection takes place.

Q5: Get supplier numbers and status for all suppliers in Paris in descending order of status (photo DB2).

```
--------> SUPPLIER          USE SUPPLIER
--------> SUPNUM            SORT ON STATUS TO NEW1 DESCENDING
--------> STATUS            USE NEW1
--------> =                 LIST SUPNUM, STATUS FOR CITY = 'Paris'
--------> CITY
..... 'Paris'
--------> DESC
--------> STATUS
--------> ENTER
```

In the above query the relation SUPPLIER is sorted in descending order according to the STATUS value, and the content is stored in a new database file called NEW1. Then, the output result is projected out of the file NEW1 under the specified condition.

This technique is unlike the one used in SQL, where the sorting occurs on the projected output result rather than on the entire relation.

Q6: For each part supplied, get part numbers and names of all cities supplying that part (photo DB3).

```
--------> SUPPLIER
--------> SHIPMENT                 USE SUPPLIER
--------> PARTNUM                  SELECT SECONDARY
--------> CITY                     USE SHIPMENT
--------> =                        JOIN TO NEW FOR P.SUPNUM = SUPNUM FIELD
--------> SUPNUM (in SUPPLIER )    PARTNUM, CITY
--------> SUPNUM (in SHIPMENT)     USE NEW
--------> ENTER                    LIST PARTNUM, CITY
```

The above query shows a join operation between the two relations SHIPMENT and SUPPLIER to generate the third relation NEW, which contains the only attributes that have to be projected.

The join operation in DBASE-II is an inefficient operation, and the technique used is not as sophisticated as other techniques used in other DBMS, where the natural or equijoin is used. The join operation in DBASE-II combines two database files (relations) in two different areas (primary and secondary area) to create a third database file. This operation takes a great deal of time and space to complete, especially when the two database files are quite large. Each record of the primary file must be compared with every record of the secondary file. this method requires a great deal of file activity. In addition to this technique, the 65, 535 records limit in any single database file may halt the join operation before its completion. For instance, if two files with 1000 records each would create a join database file with 1000, 000 records if the join expression always true, which is out of the record length limit.

Q7: Get supplier names for suppliers who supply part P2.

```
--------> SHIPMENT                    USE SHIPMENT
--------> SUPPLIER                    SELECT SECONDARY
--------> SUPNAME                     USE SUPPLIER
--------> =                           JOIN TO NEW FOR P.SUPNUM = SUPNUM
--------> SUPNUM (in SUPPLIER)        USE NEW
--------> SUPNUM (in SHIPMENT)        LIST SUPNAME FOR PARTNUM = 'P2'
--------> AND
--------> =
--------> PARTNUM
..... 'P2'
--------> ENTER
```

In the above query, NEW is the database file which contains all the attributes that are resulted from the join operation between SHIPMENT and SUPPLIER. The condition PARTNUM = 'P2' was applied on the file NEW to project SUPNAME.

Q8: Get the total number of suppliers currently supplying parts.

```
--------> SHIPMENT          USE SHIPMENT
--------> *                 COUNT
--------> COUNT
--------> ENTER
```

The answer of the above query will be an integer representing the total number of records in the file SHIPMENT.

Q9: Get the total number of shipment which are greater than 100.

```
--------> SHIPMENT          USE SHIPMENT
--------> *                 COUNT FOR QTY > 100
--------> COUNT
--------> >
--------> QTY
..... 100
--------> ENTER
```

Q10: Get the total supplied quantity of part P2 (photo DB4).

```
--------> SHIPMENT          USE SHIPMENT
--------> QTY               SUM QTY FOR PARTNUM = 'P2'
--------> SUM
--------> =
--------> PARTNUM
..... 'P2'
--------> ENTER
```

Q11: For all parts, get the part number and the weight of the part in grams. (weights are given in relation PART in pounds).

```
--------> PART              USE PART
--------> PARTNUM           LIST PARTNUM,WEIGHT*454
--------> WEIGHT
--------> *
..... 454
--------> ENTER
```

Q12: Get all part numbers and the quantity of each part divided by 10, supplied by S2 (photo DB5).

```
--------> SHIPMENT          USE SHIPMENT
--------> PARTNUM           LIST PARTNUM, QTY/10 FOR SUPNUM = 'S2'
--------> QTY
--------> /
..... 10
--------> =
--------> SUPNUM
..... 'S2'
--------> ENTER
```

1.1. Updating operations

1. UPDATE

This command is used to change the value of attributes in a database files. It is equivalent to REPLACE command in DBASE-II.

Q13: Change the colour of part P2 to red and set its city to Bath (photo DB6).

```
--------> UPDATE
--------> PART
--------> COLOUR          USE PART
--------> CITY            REPLACE ALL COLOUR WITH 'Red',
--------> =               CITY WITH 'Bath' FOR PARTNUM = 'P2'
--------> PARTNUM        LIST
..... 'P2'
--------> ENTER


ENTER NEW COLOUR: 'Red'
-----------------------
ENTER NEW CITY: 'Bath'
----------------------
```

2- **DELETE, DROP**

DELETE or DROP command may be used to delete a database file from the database and release its space.

Q14: Delete all part.

```
--------> DELETE
--------> PART              DELETE FILE PART
--------> ENTER
 OR
--------> DROP
--------> PART              DELETE FILE PART
--------> ENTER
```

A number of specified records can be deleted from the file by extending the DELETE command with specified conditions.

Q15: Delete supplier S2 (photo DB7).

```
--------> DELETE
--------> SUPPLIER
--------> =                 DELETE SUPPLIER FOR SUPNUM = 'S2'
--------> SUPNUM           PACK
..... 'S2'                 LIST
--------> ENTER
```

PACK means; release the spaces of the deleted records.

3- **DEFINE**

DEFINE is the command which is usually used to define views in a relational database. In DBASE-II there is no DEFINE command to define a relation in view concept, but there is a COPY command which is used to copy a base relation to another new relation according to certain specifications. This new relation behaves like any other base relation and can be used without restrictions. In this implementation of PDBIS, COPY is used as an equivalent command to DEFINE.

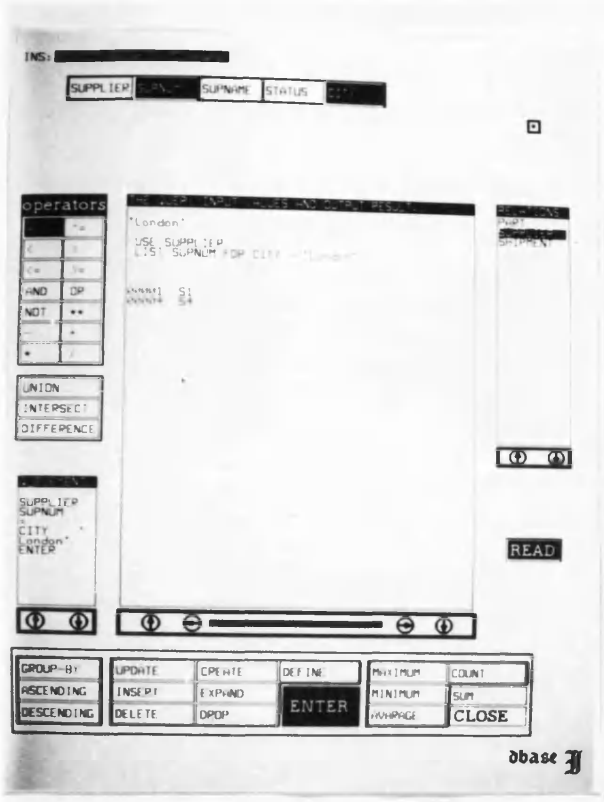
In fact, in DBASE-II any intermediate file created during the processing of the query, or any file created by using one of DBASE-II commands, is stored in the database as a database file and can be used like any other base file.

Q16: Define view of relation PART which contains part numbers, names, and cities of all part weigh more than 14 (photo DB8).

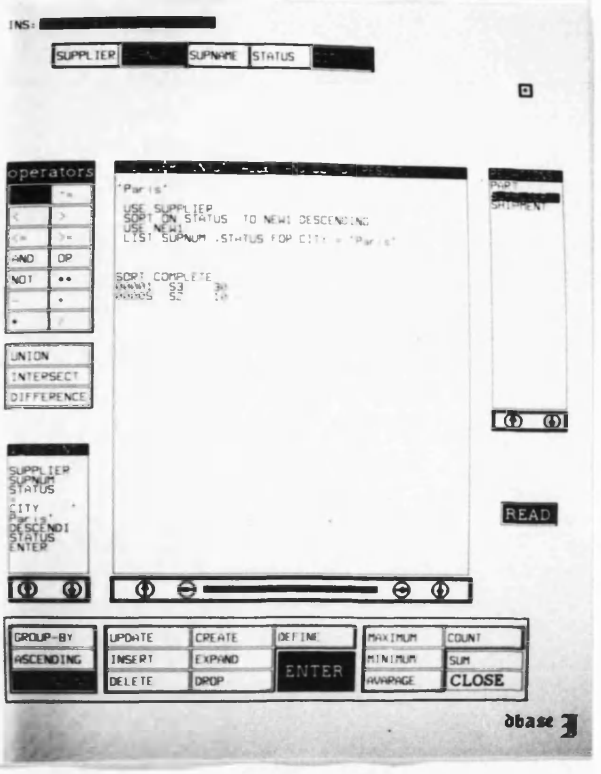
```
--------> DEFINE
--------> PART              USE PART
--------> PARTNUM           COPY PARTNUM, PNAME, CITY TO view1
--------> PNAME             FOR WEIGHT > 14
--------> CITY              USE view1
--------> >                 LIST
--------> WEIGHT
..... 14
--------> ENTER

ENTER VIEW NAME: view1
----------------------
```

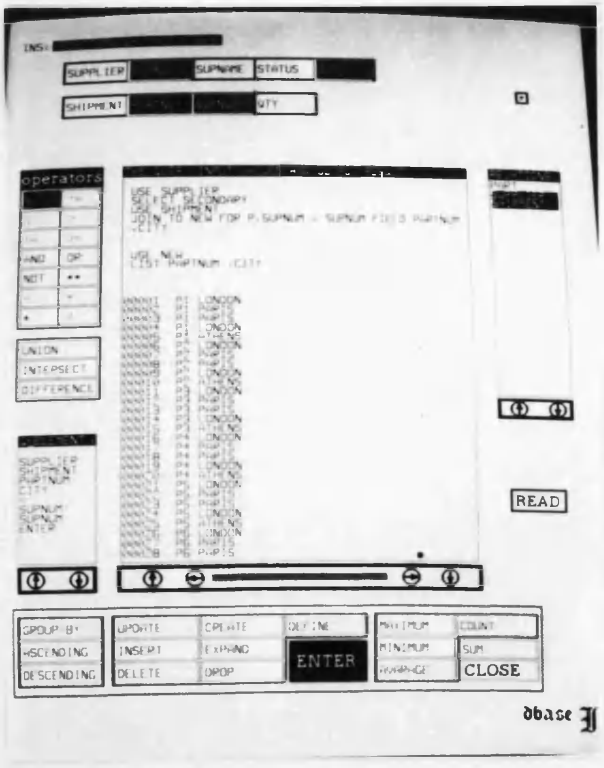
'view1' represents a view of relation PART, which contains PARTNUM, PNAME, and CITY of all the tuples in relation part which have weight > 14.



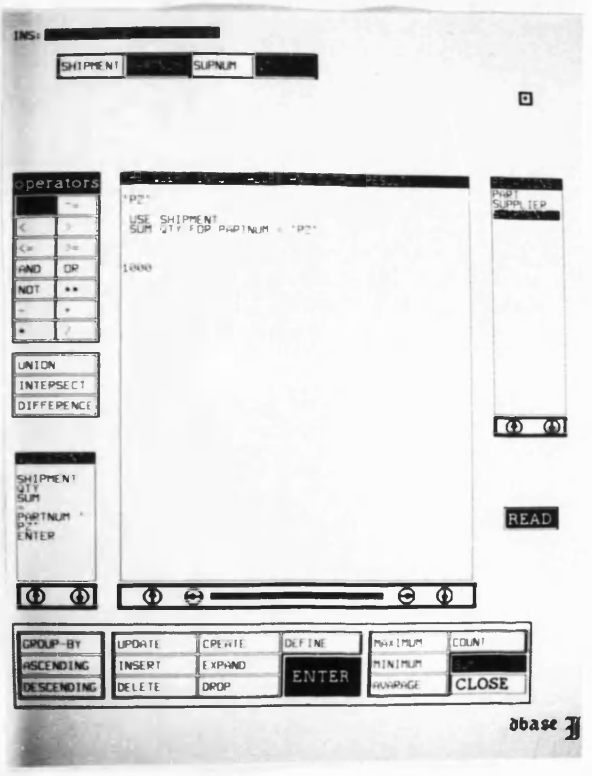
DB1



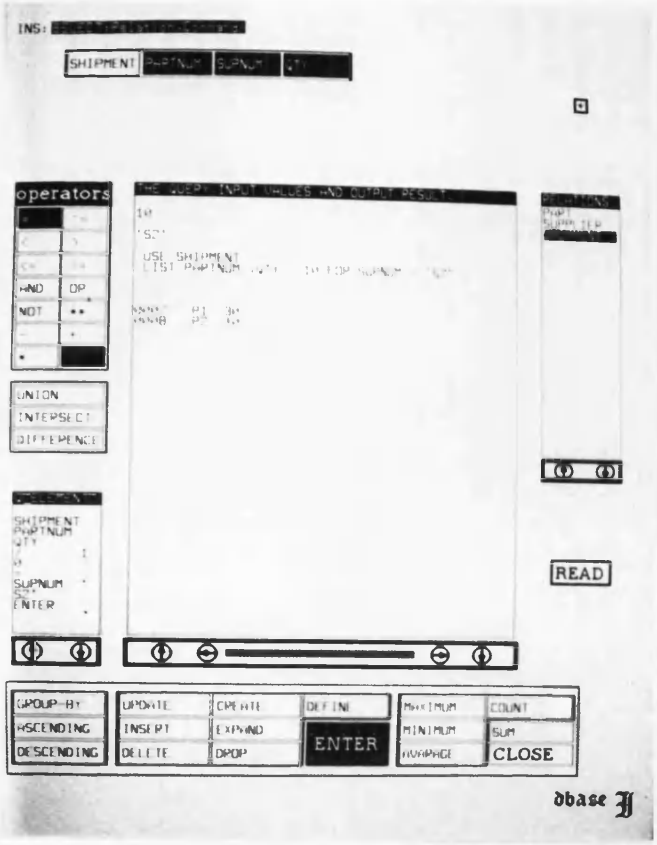
DB2



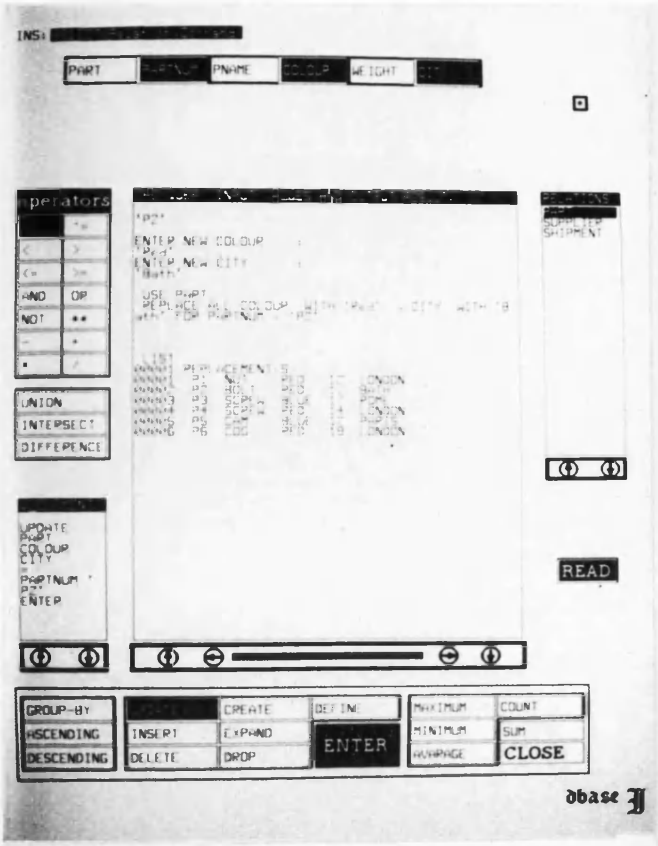
DB3



DB4

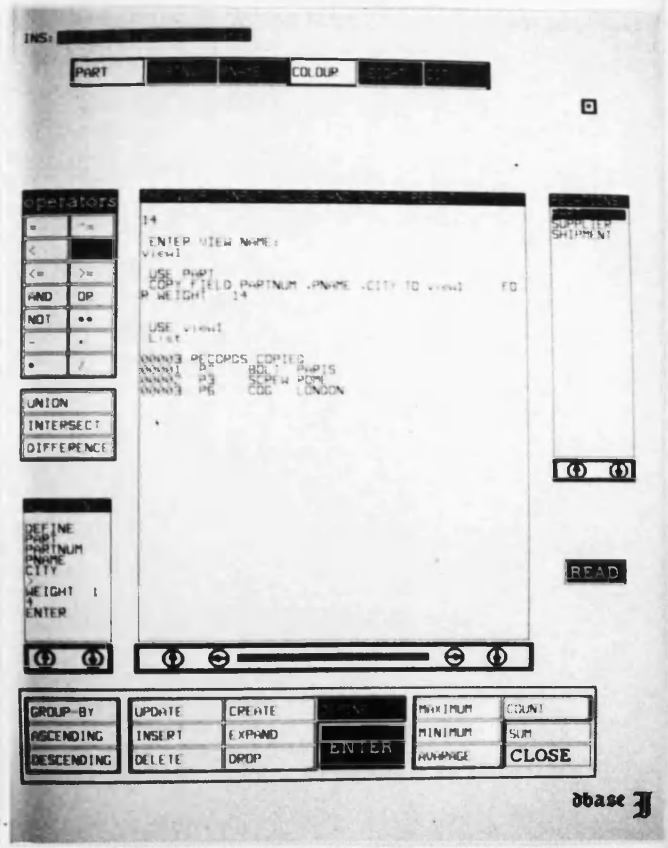


DB5



DB6

DB7



DB8

APPENDIX C

LINUS INTERFACE

1. LILA analyzer

The LILA analyzer performs the analysis of PDBIS representation of retrieving and updating queries and produces the equivalent statements in LILA representation to access a database managed and controlled by LINUS (LINU-78).

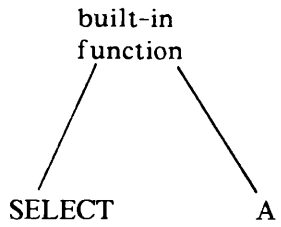
The basic component of LILA is the SELECT-FROM-WHERE block, which is used to select column values from one or more relations where tuples of the relations satisfy certain conditions. This is the same concept as SQL for building the queries. Therefore, the syntax structures of LILA is similar to the syntax structures of the SQL equivalent. Thus, the LILA analyzer depends on the same techniques which are used in SQL analyzer to analyse and translate the query.

LINUS does not provide all the utility which is available in SQL. It provides most of the utilities which are required to express a middle class query (not very complex one). ORDER-BY and GROUP-BY functions are missing in LINUS, and the creation of the database is done out of the user level, through the Multics editor by using MRDS (Multics Relational Data Store) commands.

There are three main updating functions provided by LINUS, these functions are DELETE, MODIFY, and STORE, they are implemented in the PDBIS interface as DELETE, UPDATE, and INSERT respectively. Other functions like EXPAND relation and DEFINE view are not supported by LINUS.

In addition to the usual set of built-in functions, LINUS allows the use of a set of scalar string and scalar arithmetic functions. These scalar functions have not been covered by the LILA analyzer, because they are more useful with programming languages rather than with the database manipulation languages.

The syntax structure of the built-in function in the LILA analyzer may be represented by the following binary tree.



'A' is the the argument attribute of the function.

The use of the multi-condition in WHERE clause is possible. These conditions must be separated by logical operators and may be grouped using parentheses to explicitly specify order of evaluation. The allowable logical operators in LINUS are:

& conjunction (and)
| inclusive (or)
¬ negation (not)

LILA analyzer syntax also covers; the use of set operations, UNION, INTERSECTION, and DIFFERENCES; the use of arithmetic expressions in the SELECT clause; the use of nesting SELECT-FROM-WHERE block in order to specify complex selection criteria.

2. PDBIS-LILA Interface

PDBIS-LILA interface employs most of the facilities provided by LINUS to access and update the database, except those scalar functions which have little benefit to the retrieving and updating operations.

Throughout the following examples and explanation, the characteristics of the interface are presented.

2.1. Retrieving operations

As it has been mentioned before, in the following PDBIS representation the arrows indicate the selected elements of the query on the Perq screen, while the dotted lines specify the typed constant values.

Q1: Get full details for all suppliers.

PDBIS representation

```
--------> SUPPLIER
--------> *
--------> ENTER
```

LILA representation

```
SELECT *
FROM SUPPLIER
```

This query shows how the whole relation SUPPLIER is projected.

Q2: Get supplier numbers for suppliers in Paris.

```
--------> SUPPLIER
--------> SUPNUM
--------> =
--------> CITY
..... 'Paris'
--------> ENTER
```

```
SELECT SUPNUM
FROM SUPPLIER
WHERE CITY = 'Paris'
```

This query introduced the use of the WHERE clause.

Q3: Get supplier numbers for suppliers in Paris with status < 100 (photo LILA1).

```
--------> SUPPLIER
--------> SUPNUM
--------> =
--------> CITY
..... 'Paris'
--------> AND
--------> <
--------> STATUS
..... 100
--------> ENTER
```

```
SELECT SUPNUM
FROM SUPPLIER
WHERE CITY = 'Paris'
&     STATUS < 100
```

This query shows the WHERE clause with multiple condition.

Q4: For each part supplied, get part numbers and names of all cities supplying the part.

```
--------> SHIPMENT          SELECT PARTNUM CITY
--------> SUPPLIER          FROM SHIPMENT SUPPLIER
--------> SUPNUM            WHERE SHIPMENT.SUPNUM = SUPPLIER.SUPNUM
--------> CITY
--------> =
--------> SUPNUM (in SHIPMENT)
--------> SUPNUM (in SUPPLIER)
--------> ENTER
```

This query involves a join operation between relations SHIPMENT and SUPPLIER under the join condition SHIPMENT.SUPNUM = SUPPLIER.SUPNUM.

Q5: Get all pairs of supplier numbers such that the two suppliers are located in the same city (photo LILA2).

```
--------> SUPPLIER                              SELECT F.SUPNUM  FF.SUPNUM
--------> SUPPLIER                              FROM  F:SUPPLIER  FF:SUPPLIER
--------> SUPNUM (in the first SUPPLIER)        WHERE F.CITY = FF.CITY
--------> SUPNUM (in the Second SUPPLIER)  &     F.SUPNUM ^= FF.SUPNUM
--------> =
--------> CITY (in the first SUPPLIER)
--------> CITY (in the second SUPPLIER)
--------> AND
--------> ^=
--------> SUPNUM (in the first SUPPLIER)
--------> SUPNUM (in the second SUPPLIER)
--------> ENTER
```

This query represents the selection of the same attribute from two different tuples in the same relation.

Q6: Get supplier names for suppliers who supply part P2.

```
--------> SUPPLIER          SELECT SUPNAME
--------> SUPNAME           FROM SUPPLIER
--------> =                 WHERE SUPNUM = { SELECT SUPNUM
--------> SUPNUM                    FROM SHIPMENT
--------> SHIPMENT                  WHERE PARTNUM = 'P2' }
--------> SUPNUM
--------> =
--------> PARTNUM
..... 'p2'
--------> ENTER
```

This query shows a nested SELECT-FROM-WHERE blocks.

Q7 : Get the total number of suppliers.

```
--------> SUPPLIER          COUNT { SELECT *
--------> *                           FROM SUPPLIER }
--------> COUNT
--------> ENTER
```

This query shows the use of the built-in functions. In LILA the built-in function is applied to the entire SELECT clause rather than to the attribute name as in SQL.

Q8: Get the total quantity of part P2 supplied (photo LILA3).

```
--------> SHIPMENT          SUM { SELECT QTY
--------> QTY                      FROM SHIPMENT
--------> SUM                      WHERE PARTNUM = 'P2' }
--------> =
--------> PARTNUM
..... 'P2'
--------> ENTER
```

This query involves the use of a built-in function with WHERE clause.

Q9: Get supplier numbers with status value less than the current maximum value in the relation SUPPLIER.

```
--------> SUPPLIER          SELECT SUPNUM
--------> SUPNUM            FROM SUPPLIER
--------> <                 WHERE STATUS < MAX { SELECT STATUS
--------> STATUS                              FROM SUPPLIER }
--------> SUPPLIER
--------> STATUS
--------> MAX
--------> ENTER
```

Q10: Get supplier names for suppliers who supply at least one red part (photo LILA4).

```
--------> SUPPLIER          SELECT SUPNAME
--------> SUPNAME           FROM SUPPLIER
--------> =                 WHERE SUPNUM =
--------> SUPNUM                  { SELECT SUPNUM
--------> SHIPMENT                FROM SHIPMENT
--------> SUPNUM                  WHERE PARTNUM =
--------> =                             { SELECT PARTNUM
--------> PARTNUM                        FROM PART
--------> PART                          WHERE COLOUR = 'Red' }}
--------> PARTNUM
--------> =
--------> COLOUR
..... 'Red'
--------> ENTER
```

This example shows a multi-level query (nested SELECT-FROM-WHERE blocks).

Q11: Get part numbers for parts that either weight more than 18 pounds or are currently supplied by supplier S2 (or both).

```
--------> PART              SELECT PARTNUM
--------> PARTNUM            FROM PART
--------> >                 WHERE WEIGHT > 18
--------> WEIGHT
..... 18                    UNION
--------> UNION
--------> SHIPMENT          SELECT PARTNUM
--------> PARTNUM            FROM SHIPMENT
--------> =                 WHERE SUPNUM = 'S2'
--------> SUPNUM
..... 'S2'
--------> ENTER
```

The above query shows the use of a relational operator.

Q12: For all parts get part number and weight of that part in grams (photo LILA5).

```
--------> PART                SELECT PARTNUM  WEIGHT*454
--------> PARTNUM             FROM PART
--------> WEIGHT
--------> *
..... 454
--------> ENTER
```

SELECT clause in this query contains arithmetic expression.

2.2. Update operations

1- **UPDATE**

Q13: Change the colour of part P2 to Yellow, increase its weight by 10, and set its city to 'unknown' (null) (photo LILA6).

```
--------> UPDATE              SELECT COLOUR WEIGHT CITY
--------> PART                FROM PART
--------> COLOUR              WHERE PARTNUM = 'P2'
--------> WEIGHT
--------> CITY                      MODIFY Yellow
--------> =                         MODIFY Weight+10
--------> PARTNUM                   MODIFY Null
..... 'P2'
--------> ENTER

ENTER NEW COLOUR: Yellow
------------------------
ENTER NEW WEIGHT: Weight+10
------------------------
ENTER NEW CITY: Null
--------------------
```

2- **INSERT**

Q14: Add part P7 (name 'Washer', colour 'Gray', weight 2, city 'Bath') to relation PART (photo LILA7).

```
--------> INSERT
--------> PART
--------> PARTNUM
--------> PNAME                    STORE PART P7 Washer Gray 2 Bath
--------> COLOUR
--------> WEIGHT
--------> CITY

ENTER PARTNUM VALUE: P7
-----------------------------
ENTER PNAME VALUE: Washer
-----------------------------
ENTER COLOUR VALUE: Gray
-----------------------------
ENTER WEIGHT VALUE: 2
-----------------------------
ENTER CITY VALUE: Bath
-----------------------
```

The above query adds one more tuple to the relation PART.

3- **DELETE**

Q15: Delete supplier S2.

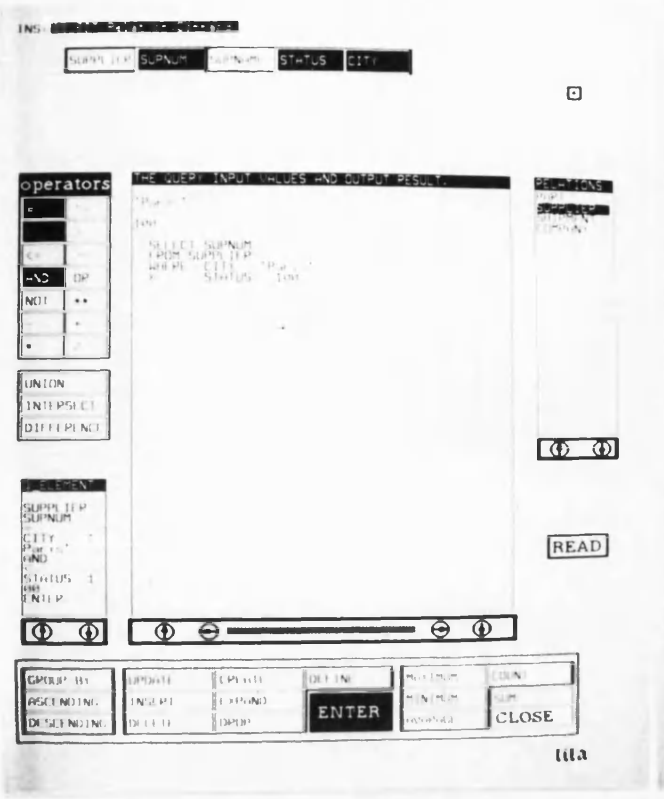
```
--------> DELETE
--------> SUPPLIER          SELECT *
--------> =                 FROM SUPPLIER
--------> SUPNUM            WHERE SUPNUM = 'S2'
..... 'S2'
--------> ENTER             DELETE
```

In the above example all tuples with SUPNUM = S2 are deleted.

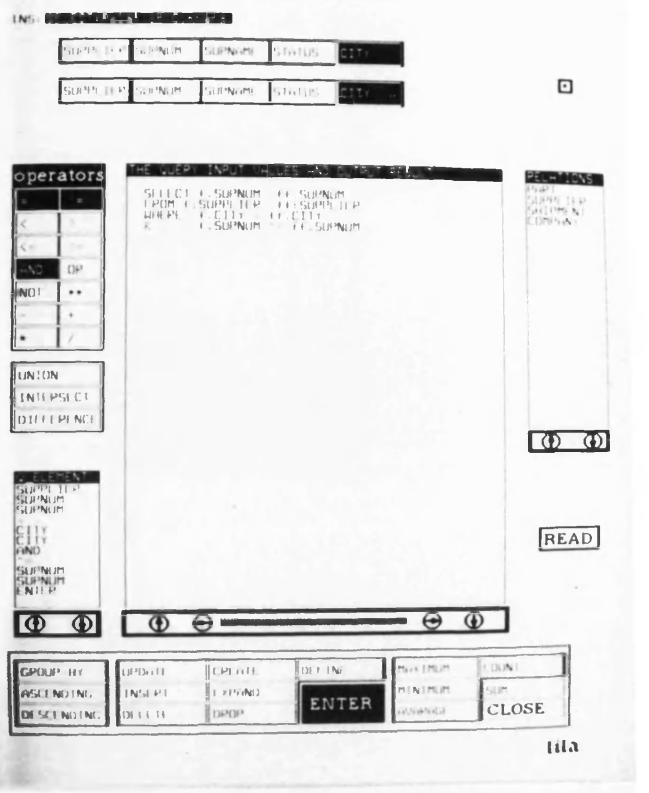
Q16: Delete all supplier.

```
--------> DELETE            SELECT *
--------> SUPPLIER          FROM SUPPLIER
--------> ENTER
                            DELETE SUPPLIER
```

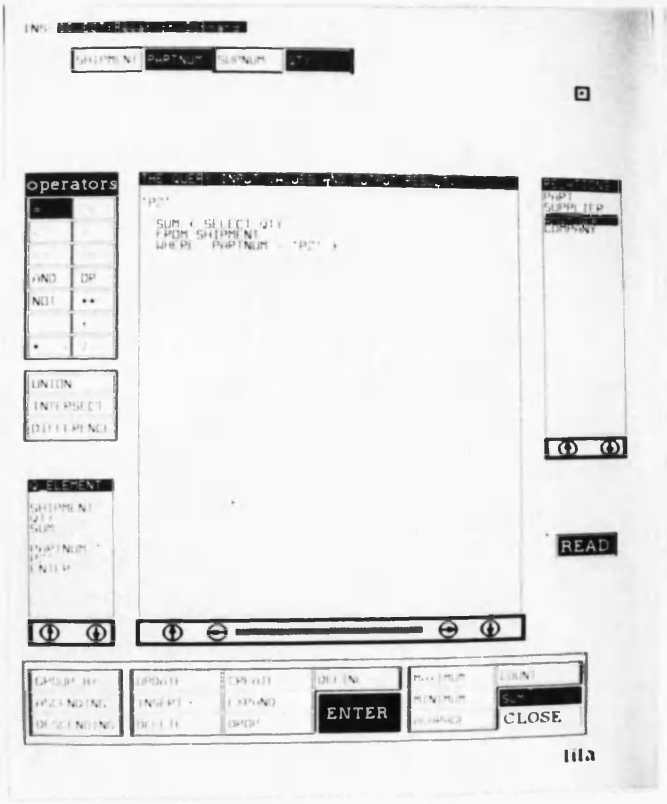
This query deletes all the tuples in the relation SUPPLIER.



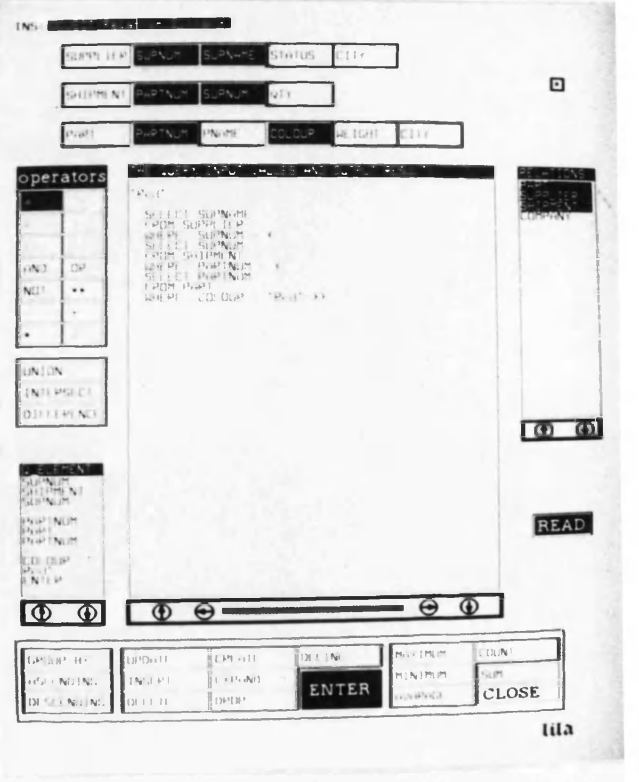
LILA1



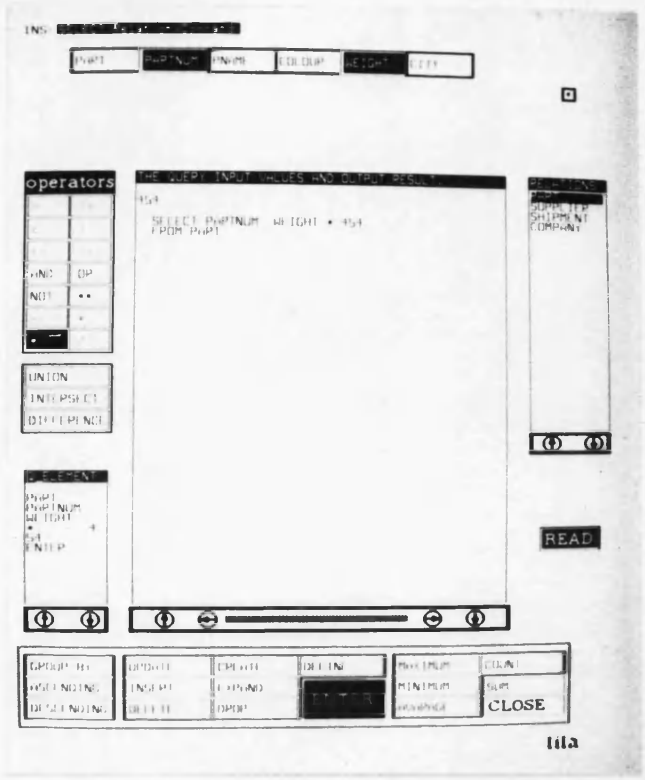
LILA2



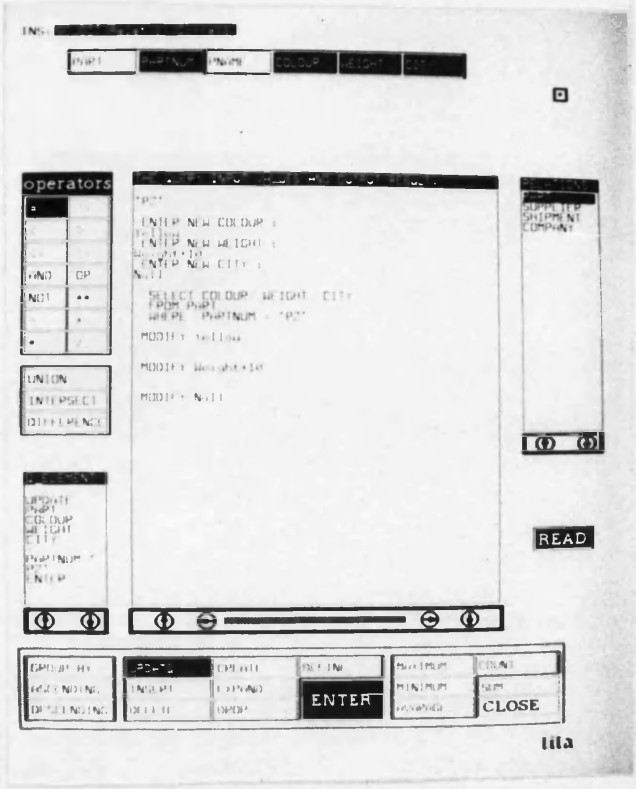
LILA3



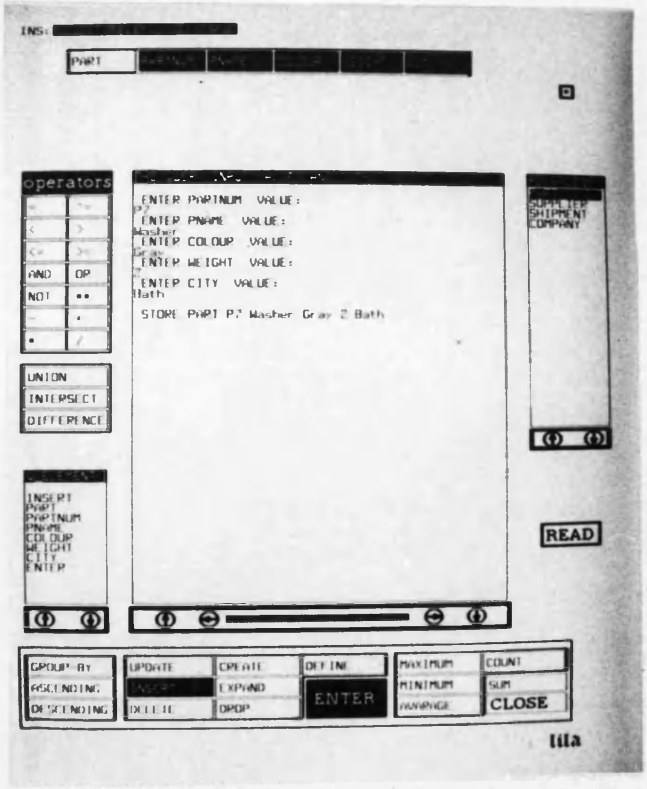
LILA4



LILA5



LILA6



LILA7

APPENDIX D

Output-File-Manager (OFM)

In this Appendix the specification of the OFM and the way it would be used is presented and explained through a number of examples.

1. OFM specification

Before the retrieved data is displayed to the user on the Perq screen, the OFM prompts the user allowing him to specify the form of the output as follows:

OUTPUT (F,R,G):

Where F stands for, that the output is to be reformatted.
R stands for, that the output is to be in form of report.
G stands for, that the output is to be in form of graph.

1.1. Formatted output

This mode of the Output-File-Manager is used to put the output result in a different format from the original one provided by the DBMS.

Example 1: Find the supplier numbers, supplier names and their status in the city of London.

PDBIS representation

--------> SUPPLIER
--------> SUPNUM
--------> SUPNAME
--------> STATUS
--------> =
--------> CITY
..... 'London'
--------> ENTER

In the above PDBIS representation of the query, the arrow indicates the selected element on the screen, while the dotted line indicates the typed constant value.

After the retrieved data are received by the OFM, the following format specification dialogue is performed to specify the required output format. In this dialogue the underlined words shows the system prompts.

```
OUTPUT (F,R,G)?: F                                    (1)
--------------------
ATTRIBUTE DISPLAY (N)?: Y                             (2)
-----------------------------
SUPNUM SUPNAME STATUS                                 (3)
--------------------------------
NEED CAPTION (Y)?:                                    (4)
----------------------
CAPTION CHANGE (N)?: Y                                (5)
--------------------------
=F1:S#, F2:SNAME                                      (6)
--
FIELD RESEQUENCE (N)?: Y                              (7)
---------------------------
=F2,F1,F3                                             (8)
--
VALUE SORT (N)?: Y                                    (9)
--------------------
=F2.ASC                                               (10)
--
OUTPUT FILE MODE (LS)?:                               (11)
-----------------------------
SAVE QUERY (N)?: Y                                    (12)
--------------------
QUERY NAME= Example 1                                 (13)
----------------
SAVE OUTPUT (N)?: Y                                   (14)
---------------------
OUTPUT NAME= Output 1                                 (15)
-----------------
```

Statement explanations:

(1) To specify the form of the output. If the user has no specific answer the output is displayed in the original format given by the DBMS, which is the default format.

(2) Question to show the selected attribute names to the user. By no reply the default value will be taken (NO).

(3) System reply by showing the selected attribute names.

(4) Question to caption the output fields. 'Y' means the caption need to be displayed before the data occurrences. 'N' is the default value for no caption.

(5) Question to change the original caption of the fields. If the reply is 'Y', the system expect the user to enter the new captions.

(6) User reply of new captions.

(7) Question to change the appearance order of the output fields. By no reply the order is not changed.

(8) The user reply for new fields order.

(9) Question to sort the output result values.

(10) Sort specification, the result will be sorted by field-2 (SUPNUM) in ascending order. There are two sort orders; ascending (ASC) and descending (DES). The default answer means no sort is required.

(11) Question to specify the output mode format. there are two output mode formats; (LS) for list format, and (L) for line format.

(12) Question to save the query.

(13) Request to enter the name under which the query is to be saved.

(14) Question to save the output.

(15) Request to enter the name under which the output is to be saved.

By having the above different options, the user can have the output in a variety of formats to satisfy the different requirements. According to the format specified in the above example the output will be displayed in the following form:

SNAME	S#	STATUS
Jones	S1	20
Clark	S2	30
Martin	S3	25
Fred	S4	30
Smith	S5	60

*End of output.

If the output file mode in statement 11 is specified by 'L', the output will be displayed in the following form:

SNAME	Jones
S#	S1
STATUS	20
SNAME	Clark
S#	S2
STATUS	30
SNAME	Martin
S#	S3
STATUS	25
SNAME	Fred
S#	S4
STATUS	30
SNAME	Smith
S#	S5
STATUS	60

*End of output.

1.2. Report writer

The report writer function provides the facility for producing a report by specifying the physical appearance of a report rather than requiring the specification of the detailed procedure that is necessary to produce that report as in the Cobol Report writer (IDS-77).

A hierarchy of levels is used in defining the logical organization of a report. Each report is divided into report groups, which in turn are divided into sequence of items. A report group contains one or more items to be presented on one or more lines. There are five report groups and one page clause which may be used to describe the appearance of the report.

1.2.1. General formats of a report description

In the following description, the braces denote alternatives, that is, one of the options in the braces must be selected, while the brackets denote an options.

(1) **Page clause format**:

The page clause is identified by 'PAGE', and used to specify the maximum number of lines in each page and the maximum number of characters in each line.

PAGE integer-1 integer-2

Integer-1, denote the number of lines per page.
Integer-2, denote the number of characters per line.

(2) **Report group format**:

$$\left\{\begin{matrix} RH \\ PH \\ DE \\ PF \\ RF \end{matrix}\right\} \quad \left\{ LINE \quad \left[\begin{matrix} integer \\ +\ integer \end{matrix}\right] \quad \left[\begin{matrix} CAPTION \\ Report-item-1 \end{matrix}\left[\ ,Report-item-2\right]\right] \right\}$$

REPORT HEADING (RH): this phrase specifies a report group that is processed only once per report to write a heading for the report in the first page.

PAGE HEADING (PH): this phrase specifies a report group that is processed on every page of the report except on the pages that are contains the report heading and the report footing.

DETAILS (DE): this phrase specifies a report group that is processed to present the output result (Details of the report) according to the specified format. The report details format is specified in the same manner as in the format mode.

PAGE FOOTING (PF): this phrase specifies a report group that is processed on all report pages except the pages specified for report heading and report footing.

REPORT FOOTING (RF): this phrase specifies a report group that is processed only once per report to write the report footing on the last page.

LINE: the 'LINE' clause must be specified to establish each print line of a report group.

Integer: the 'integer' is an absolute line number, which specifies the line number on which the print line is presented.

+integer: the '+ integer' is a line number relative to the previous line number on which a print line was presented.

CAPTION: is the clause which specify caption for a report element and has the following format;

"character string" [attribute]

Report-item : the report-item specifies a particular data item that is processed by the OFM. The report-item may take any one of the following formats:

Format (1): $\left\{\begin{matrix} DATA-ITEM \\ TIME \\ DATE-TIME \\ PNO \\ LNO \end{matrix}\right\} \left[Attribute-1\right]$ ——

These elements in this format is interpreted as follows;

'DATA-ITEM' is an attribute specified in the SELECT statement of the query. The specified DATA-ITEM will be mapped to the data value of that attribute.

DATA-ITEM ----- value.

'DATE' specifies to write the date on the report in the following format;

DATE ----- mm/dd/yy.

'TIME' specifies to write the time on the report in the following format;

$$TIME \text{ ------ } hh:tt \left\{\begin{matrix} am \\ pm \end{matrix}\right\}$$

'DATE-TIME' specifies to write the date and the time on the report in the following format;

$$DATE\text{-}TIME \text{ ------ } mm/dd/yy \ hh:tt \left\{\begin{matrix} am \\ pm \end{matrix}\right\}$$

'PNO' is a synonym for page-counter, it specifies to write the sequential page number where this specified on the report. The format of PNO is;

PNO ---- "PAGE" number.

'LNO' is a synonym for total-line-counter. It specifies to write the total line of that report, on the report. It has the format of;

LNO ---- "NUMBER" number.

$$Format\ (2): \quad \left\{\begin{matrix} AVG \\ SUM \\ CNT \\ MAX \\ MIN \end{matrix}\right\} data\text{-}item \ \left[Attribute\right]$$

This format offers a number of statistical functions, which may be used in any of the

report description group.

'AVG' Specifies to write the average of data occurrences of the data-item, on the report.

'SUM' specifies to write the total number of data occurrences of the data-item, on the report.

'CNT' specifies to write the number of data occurrences of the data-item, on the report.

'MAX' specifies to write the maximum value within every data occurrences of the data-item, on the report.

'MIN' specifies to write the minimum value within every data occurrences of the data-item, on the report.

Format(3): 'character string' [attribute]

This format specified 'character string', to be written on the report.

In the above format the data attribute can take any one of the following format:

Format(1): CAPTION ---- 'character string'

CAPTION clause specifies the caption corresponding to report item. If caption is not specified, the name of the report item becomes caption of itself.

Format(2): COL [+] integer

The COLUMN clause specifies the beginning (leftmost) position in a line for a report element.

'COLUMN Integer' specifies a display position relative to the leftmost display or printing position according to paper positioning margin setting, and/or device

characteristics.

'COLUMN + Integer' specifies a display position relative to the rightmost character of the preceding element.

In the absence of a column clause for an element, the default positioning will be +2; that is two blank characters are inserted immediately to the right of the preceding element or, for the first element of a line, the leftmost display position will be assumed 1.

Format (3): $\left\{\begin{matrix} JL \\ JR \end{matrix}\right\}$

'JL' stand for justified left, for alphanumeric data, specifies that any leading blanks within the source item or variable are to be ignored, and the leftmost nonblank character is to be aligned at the leftmost position of the space reserved for the report element.

'JR' stand for justified right, for alphanumeric data, specifies that any trailing blanks within the source item or variable are to be ignored and the rightmost nonblank character is to be aligned at the rightmost position of the space reserved for the report element.

Format(4): ID

'ID' clause specifies the output format of report items; DATA-ITEM, DATE, TIME, DATE-TIME, PNO, and LNO as they are explained previously.

1.2.2. General rules

(1) The page clause must appears in any report description, and should be the first clause to be specified.

(2) The report groups RH, PH, DE, PF, and RF may be written in any order. And any one of them may appear only in the first description statement of that description group.

Example 2: Get supplier numbers, supplier names and their status in the city of Paris.

PDBIS representation

```
--------> SUPPLIER
--------> SUPNUM
--------> SUPNAME
--------> STATUS
--------> =
--------> CITY
..... 'Paris'
--------> ENTER

OUTPUT (F,R,G)?: R
------------------
ATTRIBUTE DISPLAY (N)?:
-----------------------------
CAPTION (Y)?:
----------------
CAPTION CHANGE (N)?:
---------------------------
FIELD RESEQUENCE (N)?:
---------------------------
VALUE SORT (N)?:
--------------------
OUTPUT MODE (LS)?:
------------------------
SAVE QUERY (N)?:
--------------------
SAVE OUTPUT (N)?:
-----------------------

Enter report description:
-------------------------
=PAGE 26 72
=RH LINE 4 'REPORT-NO-1' COL 30 JL
=  LINE 'DATABASE: SUPPLIER-PART' COL 30 JL
```

```
=  LINE+2 'ISSUED BY AL-RAWI' COL 30 JL
=  LINE+5 'DATE OF THE REPORT' COL 5 DATE JL
=  LINE 'TIME OF THE REPORT' COL 5 TIME JL
=PH LINE 2 PNO COL 50 ID
=  LINE DATE-TIME COL 25 ID
=  LINE+4 CAPTION
=DE LINE SNUMBER, SNAME, STATUS
=RF LINE+10 '*** SUMMARY ***' COL 5
=  LINE 'NO-OF-SUPPLIER=' COL 5 CNT SNUMBER JL
=  LINE 'TOTAL STATUS=' COL 5 SUM STATUS JL
=  LINE 'END OF REPORT' COL 5
=  LINE+6 'NEXT REPORT WILL BE ON 20-MAR-87' COL 10
=c/r
```

According to the above specification the following report form is generated.

REPORT NO-1
DATABASE SUPPLIER-PART

ISSUED BY AL-RAWI

DATE OF THE REPORT 10/02/86
TIME OF THE REPORT 10:10 am

SUPNUM	SUPNAME	STATUS
S1	Jones	20
S2	Clark	30
S3	Martin	25
S4	Fred	30
S5	Smith	60

* End of output

SUMMARY
NO OF SUPPLIER = 5
TOTAL STATUS= 165
END OF REPORT

NEXT REPORT WILL BE ON 20-MAR-87

Example 3: Find part numbers, supplier numbers and quantity of all supplied parts.

PDBIS representation

```
------> SHIPMENT
------> S#
------> P#
------> QTY
------> ENTER

OUTPUT (F,R,G) ?: R
---------------
FIELD ATTRIBUTE DISPLAY (N) ?: Y
----------------------------
S# P# QTY
---------
NEED  CAPTION CHANGE (N) ?: Y
-----------------------------
=F1:SUPNUM, F2:PARTNUM, F3:QUANTITY
--
RESEQUENCE (N) ?: Y
---------------
=F2,F1,F3
--
VALUE SORT (N) ?: Y
-------------------
=F1 DES
--
OUTPUT MODE (LS) ?: L
-----------------
SAVE QUERY (N) ?:
---------------
SAVE OUTPUT (N) ?: Y
---------------
OUTPUT NAME= Report-2
------------

ENTER REPORT DESCRIPTION:
-------------------------

=PAGE 26 72
=PH LINE 1 DATE-TIME COL 20 ID
=PNO COL 50 JL ID
=  LINE+5 CAPTION
=DE PARTNUM, SUPNUM, QUENTITY
=PF LINE 'NUMBER OF SUPPLIER' COL 5 CNT SNUMBER JL
=PF LINE 'TOTAL OF QUANTITY=' COL 5 SUM QUENTITY JL
=c/r
```

Then the following form of report is displayed:

DATE 10/02/86 10:30 am PAGE 1

P#	P4
S#	S3
QUANTITY	10
P#	P3
S#	S2
QUANTITY	30
P#	P2
S#	S1
QUANTITY	20
P#	P2
S#	S2
QUANTITY	30
P#	P1
SUPNUM	S4
QUANTITY	30

*End of output.

NUBER OF SUPPLIER= 5
TOTAL OF QUANTITY= 120

1.3. Graph display

The Output-File-Manager allows the output to be displayed as an X-Y graph. Arithemetic operations may be applied to the graph parameters to allow the user to specify various scaling factor.

Example 4: Display the X-Y plotting between supplier status and the quantity supplied.

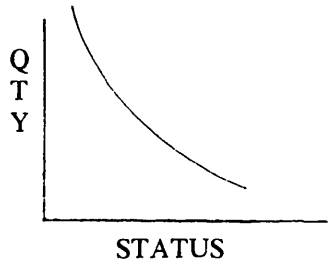
--------> SHIPMENT
--------> SUPPLIER
--------> STATUS
--------> QTY
--------> =
--------> SUPNUM (in SUPPLIER)
--------> SUPNUM (in SHIPMENT)
--------> ENTER

OUTPUT (F,R,G): G

X-axis= STATUS*10

Y-axis= QTY*10

Then the graph is displayed as:



*End of output.

REFERENCES

ANSI-75 ANSI/X3/SPARK Study Group on Database Management Systems Report.
FDT(ACM SIGMOD Bulletin), V.7, N.2, 1975.

ASHT-81 DBASE-II user manual, 1982.
ASHTON-TATE,
9929 West Jeferson Boulevard,
Culver City, Calif. 90230.

ASTR-75a M.M.Astrahan, D.D.Chamberlin.
Implementation of a structure query language.
Comm. ACM, V.18, N.10, 1975, pp 580-588.

ASTR-75b M.M.Astrahan, R.A.Lorie.
SEQUEL-XRM, a relational system.
Proc. ACM Pacific Regional Conf., April 1975, pp 34-38.

ASTR-76 M.M.Astrahan, *et al.*
System-R relational approach to database management.
ACM Trans. on Database Syst. V.1, N.2, 1976, pp 97-137.

ASTR-79 M.M.Astrahan, *et al.*
System-R, a relational database management system.
IEE Comp. Society: Computer, V.12, N.5, May 1979, pp 42-48.

ATKI-81 M.Atkinson.
Infotech state of the art report on database, series 9, N.8, 1981.
Ed. M.Atkinson.

BABB-79 E.Babb.
Implementing a relational database by means of specialized hardware.
ACM Trans. on Database Syst., V.4, N.1, March 1979, pp 1-29.

BIDM-84 C.Bidmead.
dBASE III.
Practical Computing, November 1984, pp 72-75.

BJOR-73 D.Bjorner, E.F.Codd, K.Deckert, L.L.Traiger.
The GAMMA-0 n-ary relational database interface specifications of objects and operations.
Research report, N. RJ1200, 1973, IBM Research, San Jose, Calif.

BOGU-83 B.K.Boguraev, K.S.Jones.
How to drive a database front end using general semantic information.
Proc. of the Conf. on applied natural language processing, pp 81-88.
1-3, February 1983, Santa Monica, Calif.

BOYC-75 R.F.Boyce, D.D.Chamberlin, W.F.King, M.M.Hammer.
Specifying queries as a relational expression: SQUARE.
Comm. ACM, V.18, N.11, November 1975, pp 621-628.

BUNE-79 O.P.Buneman, R.E.Frankel.
FQL-a functional query language.
Proc. of the ACM SIGMOD Int. Conf. on Management of Data,
Boston, May 30-June 1.
ACM New York, 1979, pp 52-57.

CARD-73 A.F.Cardenas.
Evaluation and selection of file organization.
Comm. ACM, V.16, N.9, 1973, pp 540-548.

CHAM-74 D.D.Chamberlin, R.F.Boyce.
SEQUEL a structured English query langusge.
ACM SIGMOD Workop on Data Description, Access and
Control, May 1974, pp 249-264.

CHAM-76 D.D.Chamberlin *et al.*
SEQUEL 2: A unified approach to data definition,
manipulation, and control.
IBM J. Res. and Dev., V.20, N.6, November 1976,
pp 560-577.

CODA-78 CODASYL Data Description Language Committee.
Journal of Development, 1978.
Pub: Secretariat of the Canadian Government.
EDP standard committee.

CODD-70 E.F.Codd.
A relational model of data for large shared data banks.
Comm. ACM, V.13, N.6, June 1970, pp 377-387.

CODD-71a E.F.Codd.
A database sublanguage founded on the relational calculus.
ACM SIGFIDENT Workshop on Data Description, Access and
Control, May 1971, pp 35-67.

CODD-71b E.F.Codd.
Relational completeness of database sublanguages.
Database Systems, Courant Computer Science Symposia 6,
Prentice-Hall, 1971, pp 65-96.
Ed., R.Rustin.

CODD-71c E.F.Codd.
Further normalization of the database relational model.
Database Systems, Courant Computer Science Symposia 6,
Prentice-Hall, 1971, pp 33-64.
Ed. R.Rustin.

CODD-71d E.F.Codd.
Normalized Database Structure: Abrief Tutorial.
Proc. ACM SIGFIDENT Workshop on Data Description,
Access and Control, May 1971, pp 1-17.

CODD-74 E.F.Codd.
Seven Steps to RENDEZVOUS with the casual user.
Database management Systems, North Holland,
April 1974, pp 179-200.
Ed. J.W.Klimble and K.L.Koffeman.

CODD-81 E.F.Codd.
Edgar Codd on SQL/DS.
Computerworld UK, 4 March 1981, pp 22-27.

CUFF-78 R.N.CUFF.
Database query systems for casual user.
M.Sc thesis 1978, University of Essex.

CUFF-81 R.N.CUFF.
The end user interface.
Infotech state of the art report on Database, series 9, N.8, 1981.
Ed. M.Atkinson, pp 252-273.

CZAR-75 B.Czarnik, S.Schoster, D.Tsichritzis.
ZETA: A relational database management system.
Proc. ACM Pacific Regional Conf., April 1975, pp 21-25.

DATE-81 C.J.Date.
An Introduction to Database System.
Third Edition, Addison-Wesley, 1981.

DBTG-69 CODASYL Data Base Task Group Report.
Pub. ACM, October 1969.

DBTG-71 CODASYL Data Base Task Group Report.
Pub. ACM and BCS, April 1971.

DONE-78 W.C.Donelson.
Spatial management information.
Proc. ACM SIGGRAPH, 1978, pp 203-209.

ELDE-84 M.Elder.
SLAVO: a fourth-generation language for personal computers.
NCC, V.53, 1984, pp 561-566.

ELLI-82 C.A.Ellis, M.Bernal.
OFFICETALK-D: An experimental office information system.
Proc. ACM SIGOA Conf. on office information system,
June 1982, pp 131-140.

EPST-80 R.Epstein, P.Hawthorn.
Design decisions for the intelligent database machine.
NCC V.49, 1980, pp 237-241.

EUFC-79 A status report on the activities of the CODASYL End User Facilities Committee (EUFC).
ACM SIGMOD Record, V.10, N.2&3, August 1979.

FRY-76 J.P.Fry, E.H.Sibley.
Evolution of Database Management System.
Computing Surveys, V.8, N.1, March 1976, pp 7-41.

GALI-82 W.O.Galitz.
Handbook of screen format design.
Q.E.D. Information Science, Inc.
Wellesley, Massachusetts 02181-0501.

GOLD-70 R.C.Goldstein, A.J.Strand.
The MacAIMS data management system.
Proc. SIGTIDET Workshop on Data Description, Access and Control, November 1970, pp 55-63.

GOLD-83 A.Gold, D.Robson.
SMALLTALK-80, the language and its implementation.
Addison-Wesley, 1983.

GREE-78 D.Greenblatt, J.Waxman.
Databases: Improving usability and responsiveness: A study of three database query languages.
New York: Academic press 1978, pp 76-98.
Ed. B.Shneiderman.

GREE-81 R.J.Greene.
An alternative approach to distributed database updating.
NCC, V.50, 1981, pp 481-485.

GROS-83 B.J.Grosz.
TEAM: A transportable natural language interface system.
Proc. of the Conf. on applied natural language processing, pp 39-62.
1-3 February 1983, Santa Monica, Calif.

HADE-84 D.J.Haderle, R.D.Jackson.
IBM Database 2 overview.
IBM Sys. J., V.23, N.2, 1984, pp 112-125.

HALL-76 P.A.V.Hall.
Optimization of single expressions in a relational database system.
IBM J. Res. and Dev., May 1976, pp 244-256.

HARR-78 L.R.Harris.
The ROBOT System: Natural language processing a pplied to database query.
Proc. ACM Annual Conf. December 1978, pp 165-172.

HELD-75 G.D.Held, M.R.Stonebreaker, E.Wong.
INGRES-A relational Database System.
NCC, V.44, May 1975, pp 409-416.

HERO-80 C.F.Herot.
Spatial management of Data.
ACM Trans. on Database Syst., V.5, N.4, December 1980, pp 493-514.

IBM-81 SQL/Data System for VSE: A relational data system for application development.
Research report N. G320-6590, February 1981.
IBM Corp. Data processing division, White plains, New York.

ICL-82a ICL-PERQ.
PERQ GRAFIKS Reference Manual.
August 1982.

ICL-82b ICL-PERQ.
System Software Reference Manual.
August 1982.

IDS-77 IDS/II End User Language Manual.
N. KRQ03A-E, 1977.
Toshiba, 1-26-5 Toranomon, Minato-Ku,
Tokyo, 105, Japan.

INGR-84 R.Ingram.
The best database.
Which Computer, September 1984, pp 115-131.

JONG-82 S.Jong.
Designing a text Editor? The user comes first.
Byte, April 1982, pp 284-300.

JOYC-76 J.D.Joyce, N.N.Oliver.
REGIS-a relational information system with graphics and
statistics.
NCC, V.45, 1976, pp 839-844.

KAME-78 I.Kameny, J.Weiner, M.Crilley, J.Burger, R.Gates, D.Brill.
EUFID: The end user friendly interface to data management systems.
Proc. 4th Int. Conf. on very large databases, 13-15 September, 1978, pp 38-391.

KELL-71 C.H.Kellogg, J.Burger, T.Diller, K.Togt.
The CONVERSE Natural Language Information Management System:
current status and plans.
Proc. ACM Symposium on Information Storage and Retrieval,
University of Maryland, College Park, 1971, pp 33-46.

KIM-79 W.Kim.
Relational database systems.
Computing Survey, V.11, N.3, September 1979, pp 185-210.

KING-79 T.J.King.
The Design of a relational database management system for historical records.
Ph.D thesis, Univesity of Cambridge, November 1979.

KING-83 T.J.King, J.K.M.Moody.
The Design and Implementation of CODD.
Software-Practice and Experience, V.13, 1983, pp 67-78.

KUHN-67 J.L.Kuhns.
Answering question by computer, a logical study.
Report N. RM-5428-PR, Rand Corp., Santa Monica, Calif., 1967.

LACR-77a M.Lacroix, A.Pirotte.
Domain-Oriented relational language.
Proc. 3rd Int. Conf. on very large databases, October 1977, pp 370-378.

LACR-77b M.Lacroix, A.Pirotte.
ILL An English Structure Query Language for Relational Databases.
North Holland, 1977, pp 237-260.
Ed. G.M.Nijssen.

LAND-82 T.Landers, R.L.Rosenberg.
An Overview of MULTIBASE.
North-Holand, 1982, pp 153-184.
Ed. H.J.Schneider.

LEE-83 A.Lee.
A Query and Editing Language for a Message Management System.
M.Sc. Thesis, University of Toronto, 1983.

LIN-76 C.S.Lin, D.C.P.Smith, J.Smith.
The design of a rotating associative memory for relational database applications.
ACM Trans. on Database Syst., V.1, N.1, March 1976, pp 53-65.

LINU-78 MULTICS Logical Inquiry and Update System (LINUS).
Reference Manual. Honeywell, March 1978.

LORI-71 R.A.Lorie, A.J.Symonds.
A relational access method for interactive applications.
Database Systems, Courant Computer Science Symposia 6, Prentice-Hall, 1971, pp 99-124.
Ed. R.Rustin.

LORI-74 R.A.Lorie.
XRM-an extended relational memory.
Report N. G320-2096, 1974.
IBM Cambridge Scientific Center, Cambridge, Massachusetts.

MACD-75a N.Mcdonald.
CUPID: a graphics oriented facility for support of
non-programmer interactions with a database.
Ph.D thesis. University of Calif., Berkeley, December 1975.

MACD-75b N.Mcdonald, M.Stonebraker.
The friendly query language.
Proc. ACM Pacific Regonal Conf., San Francisco, April 1975, pp 127-131.

MART-73 J.Martin.
Design of man-computer dialogue.
Prentice-Hall, 1973.

MART-77 J.Martin.
Computer Data-Base Organization.
Prentice-Hall, 1977.

MOTO-84 T.Moto-oka, H.S.Stone.
Fifth-generation computer systems: a Japanese project.
IEEE Computer, V.17, N.3, March 1984, pp 6-13.

NOTL-72 M.G.Notley.
The Peterlee IS/1 System.
Report N. UKSC-18, March 1972, pp 1-20.
IBM UK Scientific Center, Peterlee, England.

OZKA-77 E.A.Ozkaraham, S.A.Schuster, K.C.Sevcik.
Performance evaluation of a relational associative processor.
ACM Trans. on Database syst., V.2, N.2, June 1977, pp 175-195.

PACT-79 PA Computer and Telecommunications.
Distributed Database Technology.
Bub. by National Computer Center, 1979.
Oxford Road, Manchester M1 7ED, England.

QADA-85 G.Z.Qadah.
Database machines: A survey.
NCC, V.54, 1985, pp 211-223.

REIS-77 P.Reisner.
Use of Psycological Experimentations as an Aid to Development
of a Query Language.
IEEE Trans. on software engineering, V.SE-3, N.3, May 1977, pp 218-229.

REIS-81 P.Reisner.
Human Factors Studies of Query Language: A Survey and Assessment.
Computing Survey, V.13, N.1, March 1981, pp 13-30.

SAME-81 A Report of the British Computer Society Query Language Group.
Query Language A unified Approach.
Ed. P.A.Samet, 1981.

SENK-76 M.E.Senko.
FORAL-LP for DIAM-II, Foral with light pen, a language primer.
Research report, N. RC6328, November 1976.
IBM Thomas J.Watson Research Center, Yorktown Height,
New York 10598.

SENK-77 M.E.Senko.
FORAL-LP- Making pointed queries with light pen.
IFIP, Information processing 77, North Holland, pp 635-640.
Ed. B.Gilchrist.

SMIT-81 J.M.Smith, P.A.Bernstein, U.Dayal, N.Goodman, T.Landers,
K.W.T.Lin, E.Wong.
MULTIBASE-Integrating heterogeneous distributed database
system.
NCC, V.50, 1981, pp 487-499.

SMIT-82a D.C.Smith, C.Irby, R.Kimball, B.Replank.
Designing the star user interface.
Byte April 1982, pp 242-282.

SMIT-82b D.C.Smith, C.Irby, R.Kimball, B.Replank.
The STAR user interface: an overview.
NCC, V.51 1982, pp 515-528.

STEU-74 J.Steuert, J.Goldman.
The relational data management system: a prespective.
ACM SIGMOD Workshop on Data Description, Access and Control,
1974, pp 295-320.

STON-76 M.Stonebraker, E.Wong, P.Kreps, G.Held.
The design and implementation of INGRES.
ACM Trans. on Database Syst., V.1, N.3, 1976, pp 189-222.

SU-75 S.SU, G.Lipovski.
CASSM: a cellular system for very large database.
Proc. 1st Int. Conf. on very large databases, September 1975.

TESL-81 L.Tesler.
The SMALLTALK environment.
Byte, August 1981, pp 90-147.

TESK-84 F.N.Teskey, N.Dixon, S.C.Holden.
Graphical interface for binary relationship database.
Information Technology, V.3, N.2, April 1984, pp 67-77.

THOM-69 F.P.Thompson, P.Lockeman,B.H.Dostert, R.Deveill.
Extensible Language System.
Proc. ACM 24th National Conf., New York, 1969, pp 399-417.

THOM-75 J.C.Thomas, J.D.Gould.
A psychological study of Query-By-Example.
NCC, V.44, 1975, pp 439-445.

TODD-76 S.Todd.
The Peterlee relational test vehicle-A system overview.
IBM Sys. J., V.15, N.4, 1976, pp 285-307.

TSIC-80 D.Tsichritzis.
OFS: An integrated form system.
IEEE Int. Conf. on very large databases, 1980, pp 161-166.

TSIC-82 D.Tsichritzis.
A form management.
Comm. ACM, V.25, N.7, July 1982, pp 453-478.

UDAG-82 Y.Udagawa, S.Oshsuga.
Novel technique to interact with relational database by
using a graphics display.
Journal of Information Processing, V.5, N.4, 1982, pp 256-264.

WALT-78 D.L.Waltz.
An English language question-answering system for a large
relational database.
Comm. ACM, V.21, N.7, Jully 78, pp 526-539.

WEBS-84 B.F.Webster.
The MACINTOSH Overview.
Byte, August 1984, pp 238-254.

WHIT-74 V.K.M.Whitney.
Relational data management information techniques.
ACM SIGMOD Workshop on Data Description, Access and Control, 1974, pp 321-348.

WILL-83 G.Williams.
The LISA computer system.
Byte, February 1983, pp 33-50.

WILS-80 G.A.Wilson, C.F.Herot.
Semantics vs. Graphics: To show or not to show.
IEEE Int. conf. on very large databases, 1980, pp 183-197.

WONG-76 E.Wong, K.Yossifi.
Decomposition-A strategy for Query Processing.
ACM Trans. on Database Syst., V.1, N.3, 1976, pp 223-241.

XERO-81 The Xerox learning research group.
The SMALLTALK-80 system.
Byte, August 1981, pp 36-48.

ZLOO-75 M.M.Zloof.
Query-By-Example.
NCC, V.44, 1975, pp 431-437.

ZLOO-76 M.M.Zloof.
Query-By-Example: Operations on Hierarchical database.
NCC, V.45, 1976, pp 845-853.

ZLOO-77 M.M.Zloof.
Query-By-Example: a database language.
IBM Sys. J., V.16, N.4, 1977, pp 324-343.

ZLOO-78 M.M.Zloof.
Design aspect of the Query-By-Example database management language.
Database Improving Usability and Responsive.
Academic press, 1978, pp 29-55.
Ed., B.Shneiderman.

ZLOO-82 M.M.Zloof.
Office-By-Example: A business language that unifies data and word processing and electronic mail.
IBM Sys. J., V.21, N.3, 1982, pp 273-304.